✨ Member-only story

GETTING STARTED

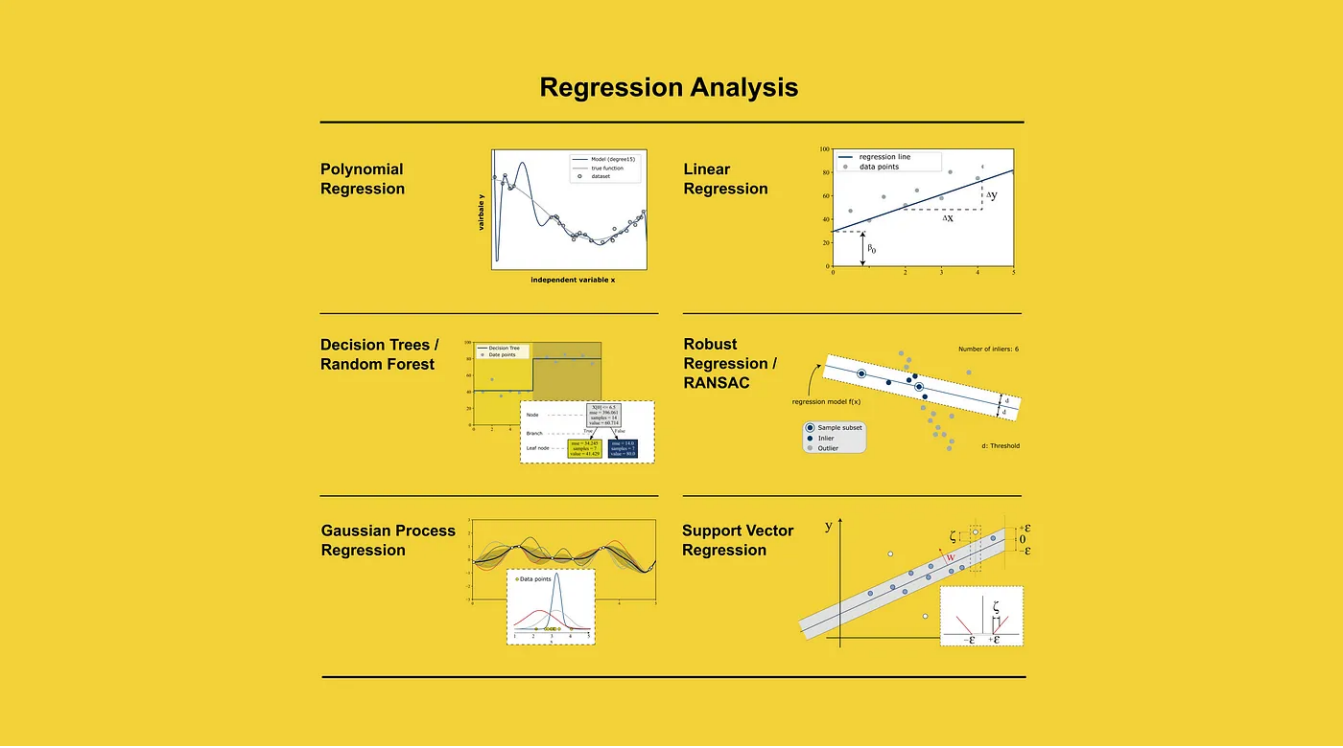
7 of the Most Used Regression Algorithms and How to Choose the Right One

Linear and Polynomial Regression, RANSAC, Decision Tree, Random Forest, Gaussian Process and Support Vector Regression

Dominik Polzer · Follow

Published in Towards Data Science

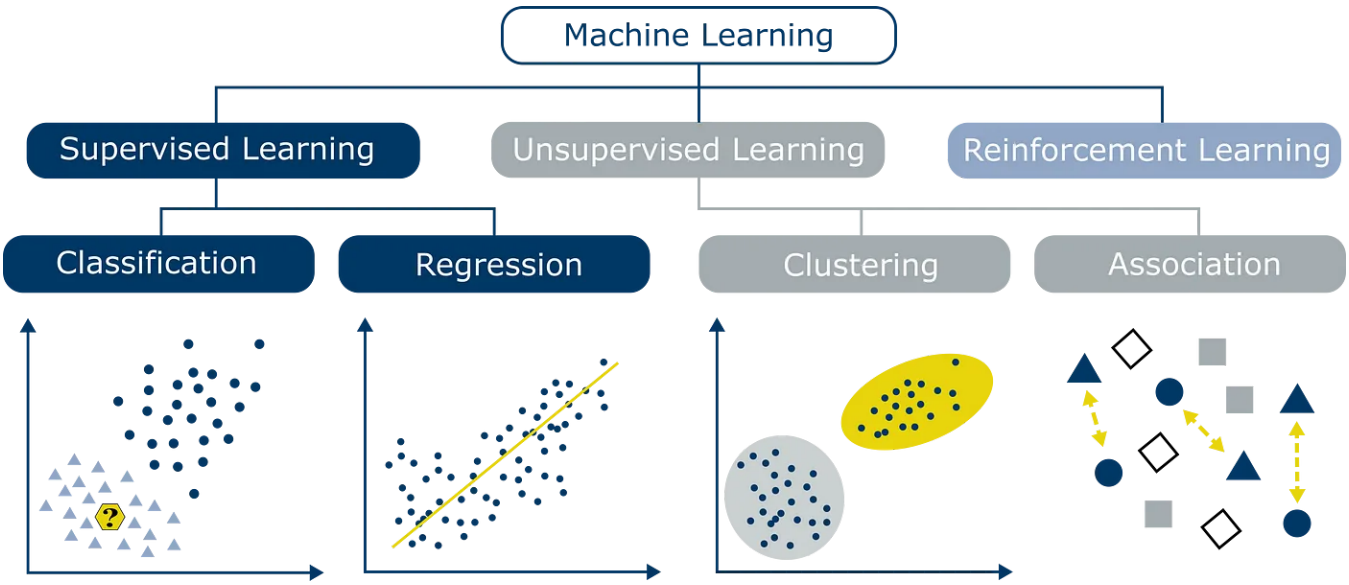
27 min read · Jun 21, 2021



Regression Algorithms — Image by the author

Regression is a subset of *Supervised Learning*. It learns a model based on a training dataset to make predictions about unknown or future data. The description '*supervised*' comes from the fact that the target output value is already defined and

part of the training data. The difference between the subcategories Regression and Classification is only due to the output value. While Classification divides the dataset into classes, Regression is used to output continuous values. [Ras16]



Overview of types of learning — Image by the author

This article introduces a few of the most used Regression methods, explains some metrics to evaluate the performance of the models and describes how the model building process works.

- **Regression methods**
  - Multiple Linear Regression
  - Polynomial Regression
  - Robust Regression — RANSAC
  - Decision Tree
  - Random Forest
  - Gaussian process regression
  - Support Vector Regression
- **Model evaluation:** How do you evaluate the generated models?
- **Model building process:** How do I find the best regression method and model for the problem at hand?

1. Regression Methods

Multiple Linear Regression

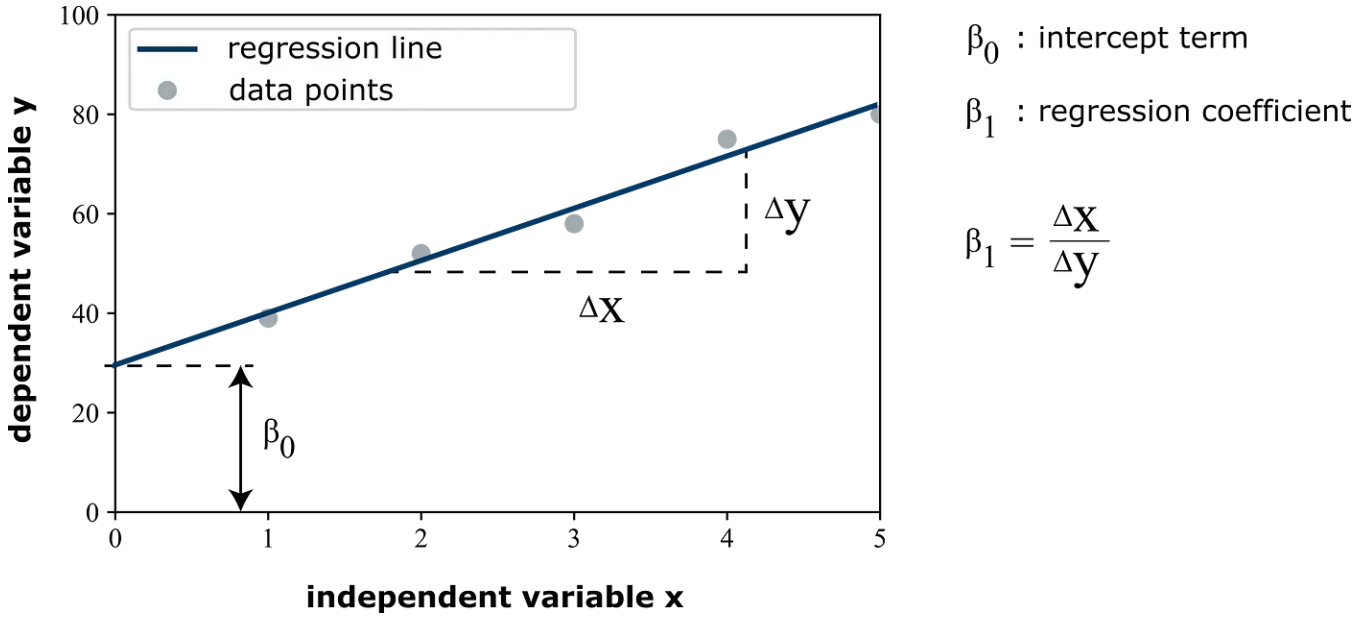
Linear regression models assume that the relationships between input and output variables are linear. These models are quite simplistic, but in many cases provide adequate and tractable representations of the relationships. The model aims a prediction of real output data `Y` by the given input data `X = (x_1, x_2, …, x_p)` and has the following form:

$$f(x) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$$

β describes initially unknown coefficients. Linear models with more than one input variable `p > 1` are called multiple linear regression models. The best known estimation method of linear regression is the *least squares method*. In this method, the coefficients $\beta = \beta\_0, \beta\_1 \ldots, \beta\_p$ are determined in such a way that the ***Residual Sum of Squares (RSS)*** becomes minimal.

$$RSS = \sum_{i=1}^{n} (y_i - f(x_i))^2 = \sum_{i=1}^{n} (y_i - \beta_0 + \sum_{j=1}^{p} X_j \beta_j)^2$$

Here, `y_i-f(x_i)` describes the residuals, β_0 the estimate of the ***intercept term***, and β_j the estimate of the ***slope parameter*** [Has09, p.44].



Linear Regression: interception term and regression coefficients — Image by the author

Polynomial Regression

By transforming the input variables, e.g. by the logarithm function, the root function etc., non-linear and polynomial relationships can be represented. Nevertheless, these are linear models, as this designation is based on the linearity of the input parameters. [Has09, p.44]

The modelling of such correlations is done using so-called trend models. If the rough course is already evident from the data, regression approaches can be specified. [Fah16, p.512] The following table shows frequently used trend models for simple linear regression.

$f(x) = \beta_0 + \beta_1 x$	Linear Trend
$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2$	Quadratic Trend
$f(x) = \beta_0 + \beta_1 x + ... + \beta_q x^q$	Polynomial Trend
$f(x) = \beta_0 \exp(\beta_1 x)$	Exponential Trend
$f(x) = \frac{\beta_0}{\beta_1 + \exp(-\beta_2 x)}$	Logistic Growth Curve

Global trend models [Fah16, p.512]

A direct function for polynomial regression does not exist, at least not in `Scikit-learn`. For the implementation the `pipeline` function is used. This module combines several transformer and estimation methods in a chain and thereby allows the fixed sequence of steps in the processing of the data. [Sci18g]

The difference to the previously known linear regression is the preliminary step. The function `PolynomialFeatures` creates a new matrix containing all polynomial combinations of the features of the input matrix `X`. [Sci18h][Sci18]

The following code snippet:

```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2)
poly.fit_transform(X)
```

preprocessing_polynomial_features.py hosted with ❤ by GitHub view raw

transforms the *input vector* `X` as follows:

$$X = \begin{pmatrix} x_{11} & x_{12} \\ x_{11} & x_{12} \end{pmatrix} \rightarrow X = \begin{pmatrix} 1 & x_{11} & x_{12} & x_{11}^2 & x_{11}x_{12} & x_{12}^2 \\ 1 & x_{21} & x_{22} & x_{21}^2 & x_{21}x_{22} & x_{22}^2 \end{pmatrix}$$

The function of the linear model is then:

$$f(x) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \beta_5 X_5 + \beta_6 X_6$$

with:
$$X_1 = [1,1]$$
$$X_2 = [x_{11}, x_{22}]$$
$$X_3 = [x_{12}, x_{22}]$$
$$X_4 = [x_{11}^2, x_{21}^2]$$
$$X_5 = [x_{11}x_{12}, x_{21}x_{22}]$$
$$X_6 = [x_{12}^2, x_{22}^2]$$

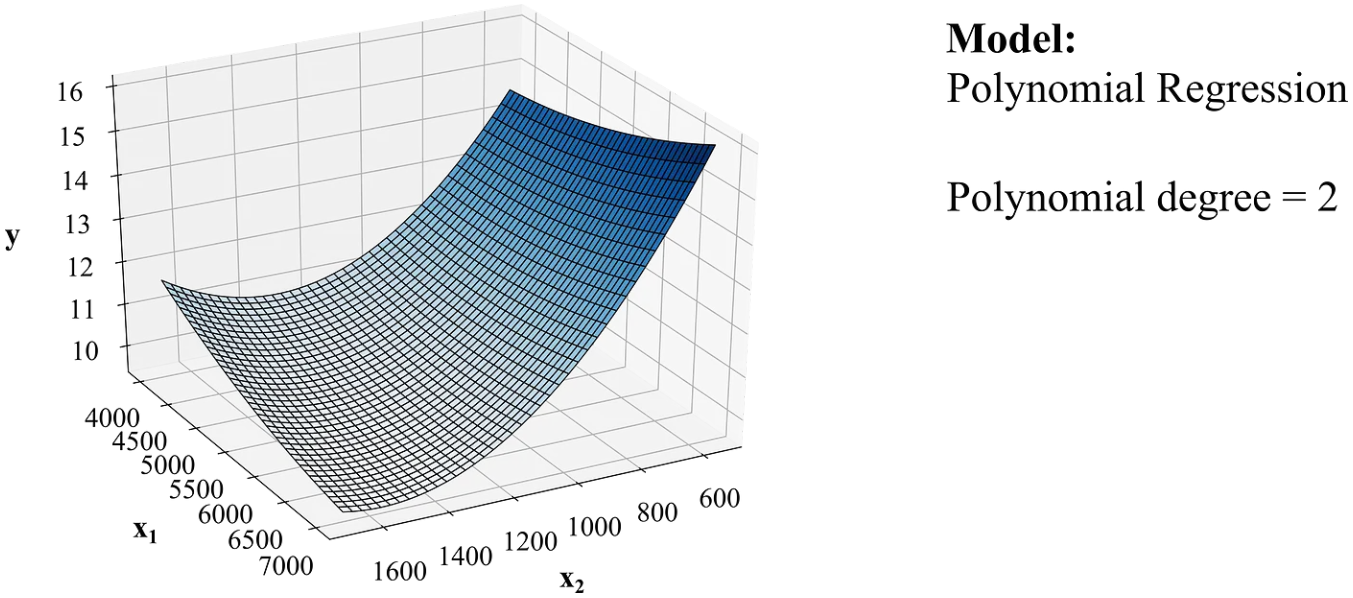
The following snippet shows the application of Polynomial Regression in `scikit-learn`. The `pipeline` function is not absolutely necessary here, the transformation of the input matrix `X` and the subsequent model building can also be executed by the corresponding commands one after the other. However, the pipeline function is required if polynomial model building is applied within the *Cross-validation* function (more in the evaluation section of this article).

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline

# define the pipeline and train model
model = Pipeline([('poly', PolynomialFeatures(degree=2)),
                  ('linear', LinearRegression(fit_intercept=False))])

model.fit(X_train, y_train)

# print out regression coefficients
print(model.named_steps['linear'].coef_)
print(model.named_steps['linear'].intercept_)
```

sklearn_polynomial_regression.py hosted with ❤ by GitHub view raw

Polynomial regression allows control of model complexity via the `polynomial degree`. Parameters that are set by the user before the algorithm is executed are called ***hyperparameters***. Most regression methods include several *hyperparameters*, which significantly influence the accuracy of the resulting regression model. You can find a explanation how to find the optimal *hyperparameters* in the section "Model evaluation".

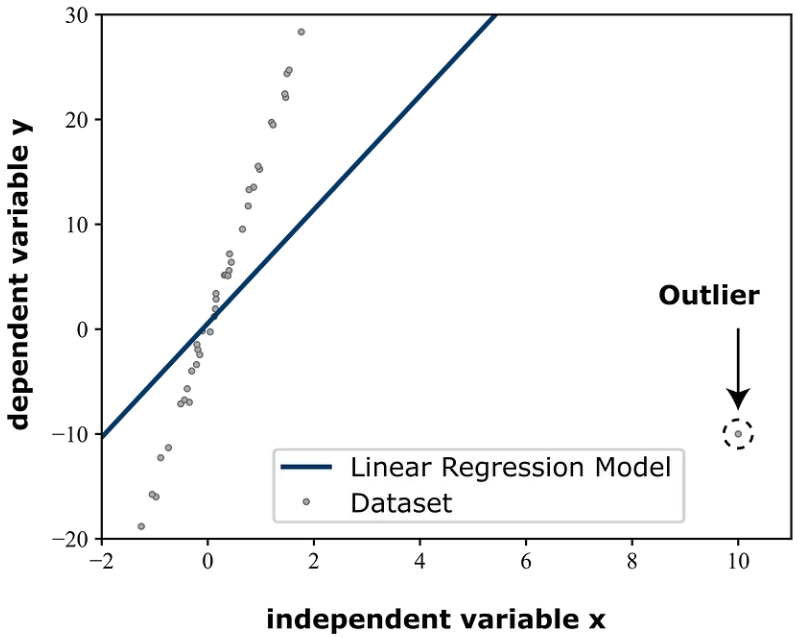
The following example shows a Polynomial Regression model with the polynomial degree 2. The model resulted from an attempt to predict the energy consumptio of a milling machine. The target value y is the energy consumption [kJ], the used attributes are the axis rotation speed [1/min] and the feed rate [mm/min].



Polynomial Regression: Sample Model — Image by the author

Robust Regression — RANSAC

Regression procedures based on *least squares estimation* are very susceptible to outliers because the variances are evaluated quadratically. The figure below illustrates the effect of a single outlier on the result of a linear regression.

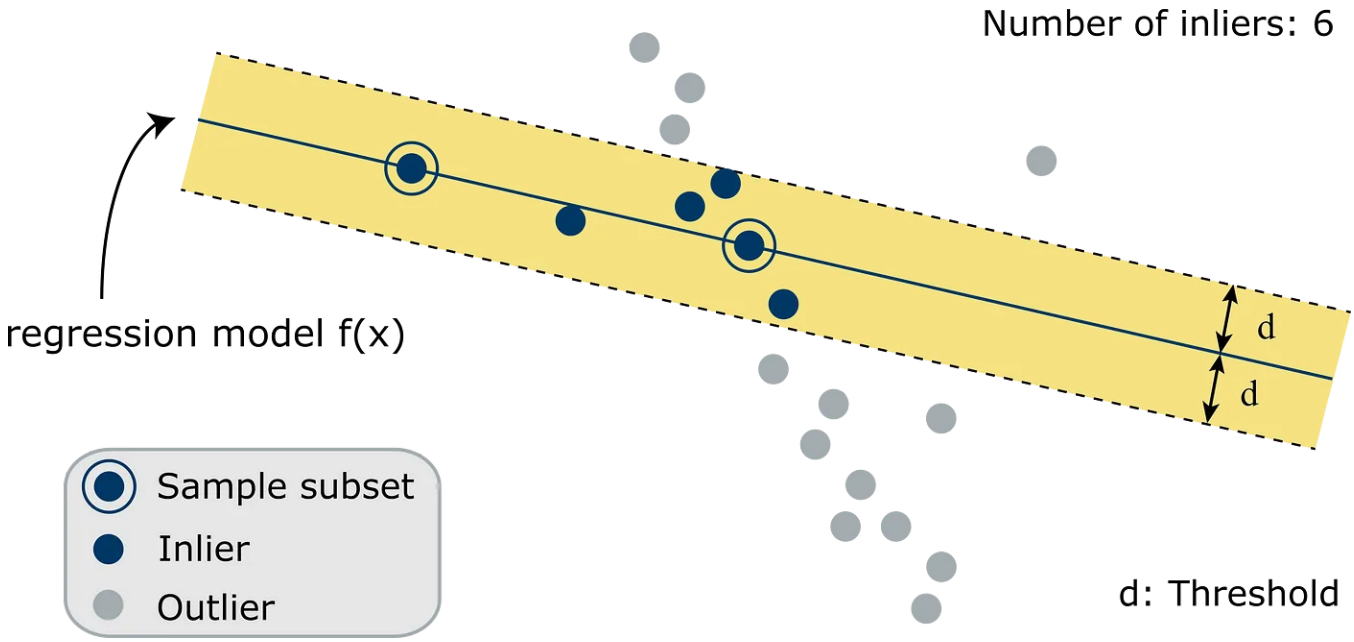


Effects of individual outliers on the linear regression model — Image by the author

Robust regression methods circumvent this weakness. The term "*robustness*" describes the ability of a static method to model distributions that do not correspond to the normal distribution. [Wie12] A measure of robustness is the so-called "*breakdown point*", which indicates the proportion of the data (e.g. Outliers) that is tolerated by the statistical method. [Hub05]

Probably the best known Robust Regression algorithm is the *Random Sample Consensus (RANSAC) algorithm,* introduced in 1981 by Martin Fischler and Robert Bolles. RANSAC is widely used in the field of machine vision. [Fis80]

The operation of the algorithm can be explained in five steps, which are executed iteratively. (1) At the beginning, the procedure selects a random sample from the data and uses it for model building. In the following figure, the ***sample*** includes the two circled data points. (2) The error of all data points to the model f(x) are then calculated and compared to the user-defined threshold. If the deviation is below this value, the data point is considered an ***inlier***.



RANSAC algorithm — Image by the author

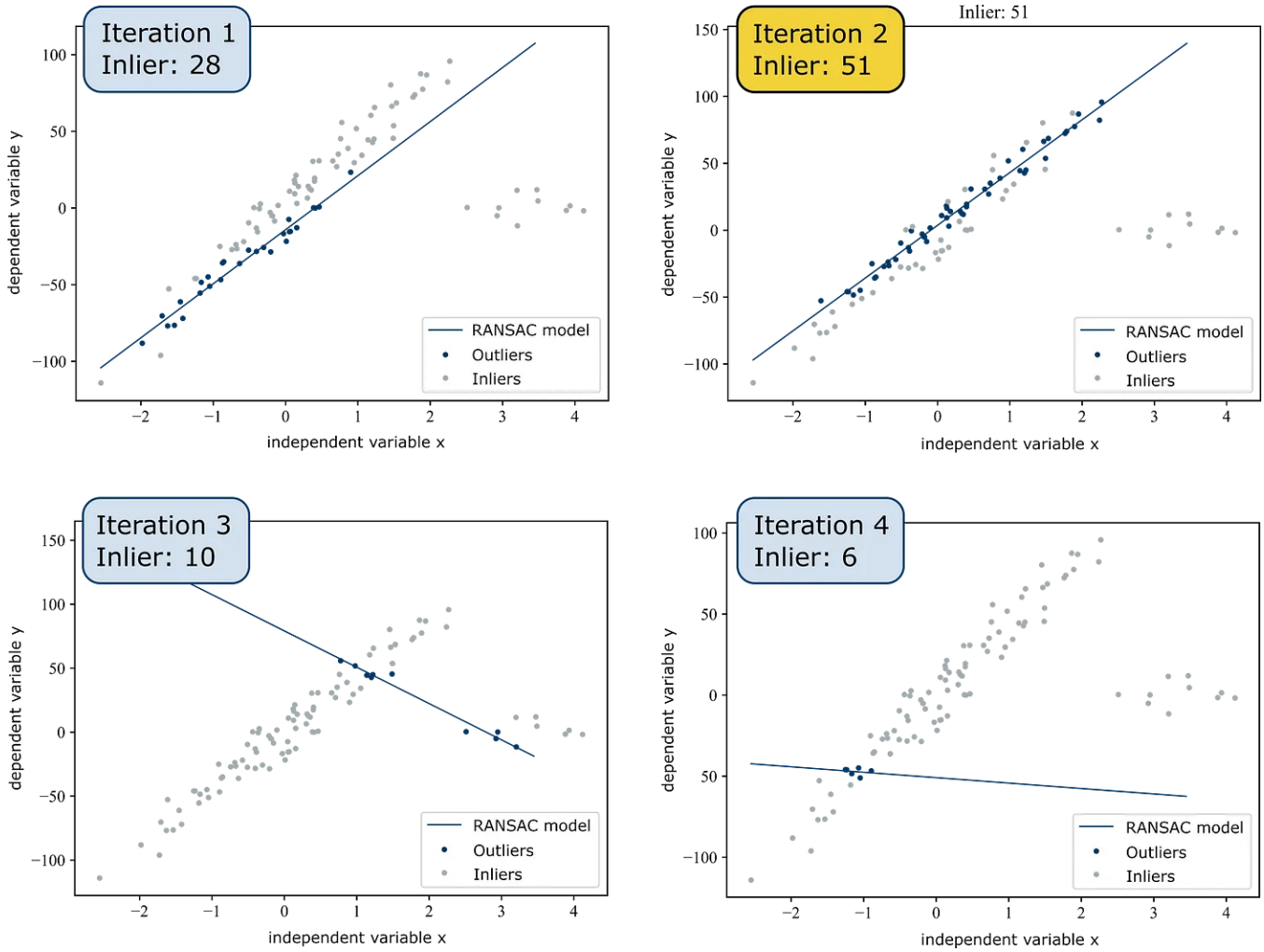
(3) This process is repeated until a specified number of iterations have been run or a specified performance of the model has been achieved. As model results is the function that yields the most *inliers*. [Ras18]

The following Python snippet describes the implementation using `scikit-learn`. The maximum number of iterations is set to four and the minimum sample size to two. These values are adjusted depending on the dataset size. For the determination of the *inliers*, the absolute values of the vertical distances between the data points and the regression line is calculated. [Ras18][Sci18b]

```
from sklearn.linear_model import RANSACRegressor

# Set RANSAC hyperparameters
ransac = RANSACRegressor(LinearRegression(),
                max_trials=4,           # Number of Iterations
                min_samples=2,          # Minimum size of the sample
                loss='absolute_loss',   # Metrics for loss
                residual_threshold=10   # Threshold
                )

# Train model
ransac.fit(X_train, y_train)
```

ransac_regression.py hosted with ❤️ by GitHub view raw

The following figure shows an example of iterative model building after executing the code snippet above. Iteration 2 shows the best performance of the model.



RANSAC algorithm: Four iterations of the model building process (Min_Samples =2, Threshhold = 20) — Image by the author

The number of iterations n required for an outlier-free subset to be selected from the data points at least once with a certain *probability* p can be determined as follows [Rod04][Dan18]:

$$n = \frac{\log\left(1-p\right)}{\log\left(1-\left(1-\delta\right)^{s}\right)}$$

with: $s =$ Number of data points of the sample
$\delta =$ Relative share of outliers in the dataset
$p =$ Probability

Assuming that the relative proportion of outliers δ is about 10%, the number of iterations `n` at which an outlier-free subset is selected at least once with a probability of `p = 99%` is calculated to:

$$n = \frac{\log\left(1 - 0,99\right)}{\log\left(1 - \left(1 - 0,1\right)^2\right)} = 3$$

Decision Trees and Random Forest

A *Decision Tree* grows by iteratively splitting tree nodes until the 'leaves' contain no more impurities or a termination condition is reached. The creation of the *Decision Tree* starts at the root of the tree and splits the data in a way that results in the largest ***Information Gain IG***. [Ras18, p.107][Aun18][Has09, p.587][May02][Sci18c]

In general, the *Information Gain IG* of a feature a is defined as follows [Qui86][Bel15, p.47][Ras18, p.107]:

$$IG(D_p, a) = I(D_p) - \sum_{j=1}^{m} \frac{N_j}{N_p} I(D_j)$$

with: N_p = Total number of instances of the parent node
N_j = Total number of instances of the j-th child node
D_p = Data amount of the parent node
D_j = Data amount of the j-th child node
I = Measure of impurity

In binary *Decision Trees*, the division of the total dataset `D_p` by *attribute* `a` into `D_left` and `D_right` is done. Accordingly, the *information gain* is defined as:

$$IG(D_p, a_i) = I(D_p) - \frac{N_{links}}{N_p} I(D_{links}) - \frac{N_{rechts}}{N_p} I(D_{rechts})$$

The algorithm aims at maximizing the *information gain*, i.e. the method wants to split the total dataset in such a way that the *impurity* in the *child nodes* is reduced the most.

While classification uses `entropy` or the `Gini coefficient` as a measure of impurity, regression uses, for example, the `Mean Squared Error (MSE)` as a measure of the

impurity of a node. [Ras18, p.347].

$$I(t) = MSE(t) = \frac{1}{N_t} \sum_{i \in D_t} (i^{(i)} - \hat{y}_t)^2$$

with: N_t = Number of training objects of the node
D_t = Training subset of the node
$y^{(i)}$ = Actual target value
$\hat{y}_t$ = Predicted target value

Splitting methods that use the `Mean Squared Error` to determine impurity are also called *variance reduction methods*. Usually, the tree size is controlled by the *maximum number of nodes* `max_depth`, at which the division of the dataset stops. [Has09, S.307]

```
from sklearn.tree import DecisionTreeRegressor

# Build decision tree
tree=DecisionTreeRegressor(max_depth=1)
tree.fit(X_train,y_train)
```

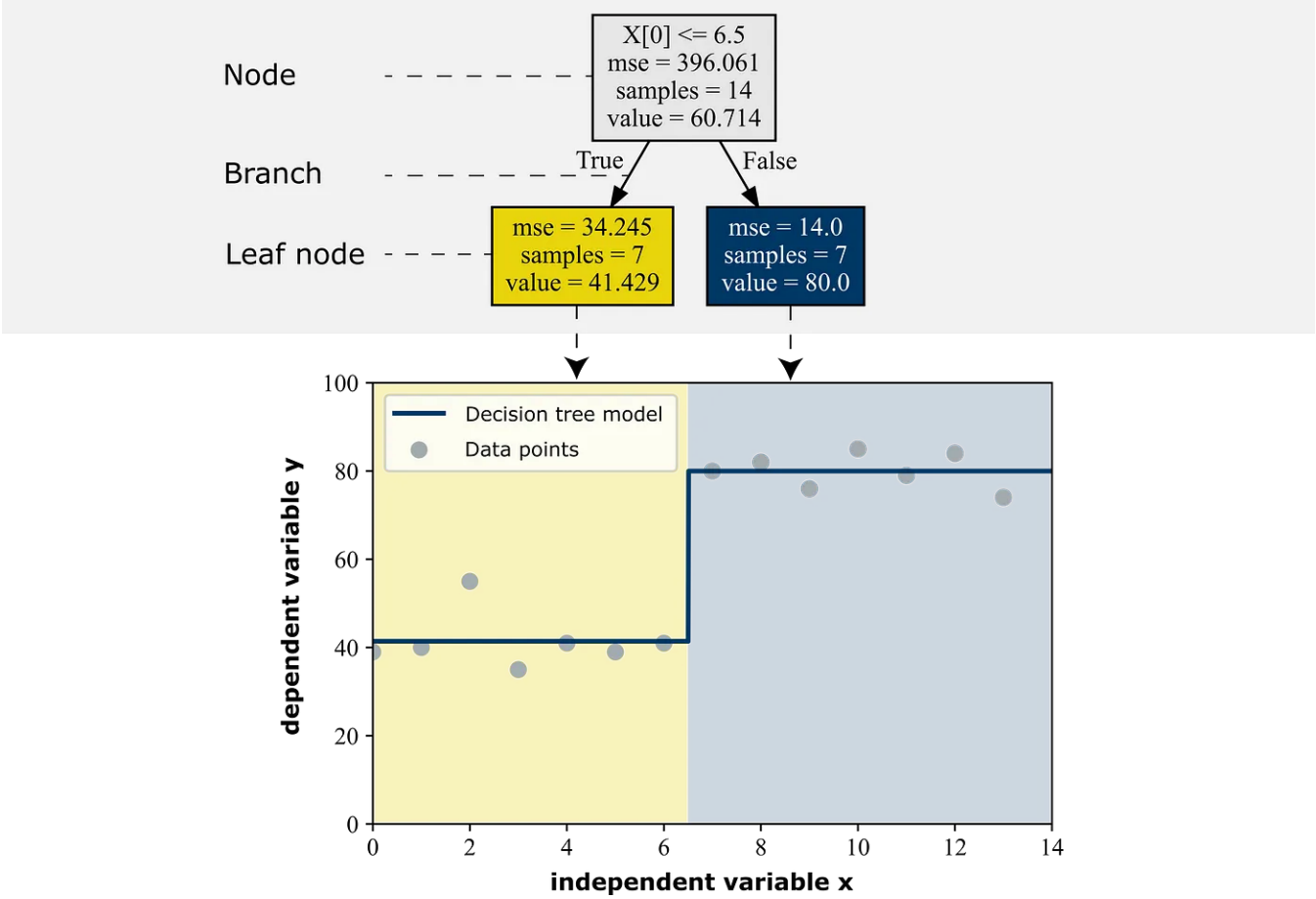
decision_tree_regressor.py hosted with ❤ by GitHub view raw

The visualization of the nodes and leaves of the decision tree can be done using the `graphviz` function:

```
import graphviz
from sklearn.tree import export_graphviz

dot_data = export_graphviz(tree, out_file=None)
graph = graphviz.Source(dot_data)
```

decision_tree_visualization.py hosted with ❤ by GitHub view raw

The following figure shows the result of the *Decision Tree* for a simple dataset. The method splits the dataset into two partial subsets (left and right) in such a way that the *variance* is reduced as much as possible. For the dataset shown, the limit at which the dataset is split the first time is 6.5.



Decision tree for a simple two-dimensional case with a depth of one — Image by the author

Random Forest

By merging several uncorrelated *Decision Trees*, often a significant improvement of the model accuracy can be achieved. This method is called *Random Forest*. The trees are influenced by certain random processes (randomization) as they grow. The final model reflects an averaging of the trees.

Different methods of randomization exist. According to Breiman, who coined the term '*Random Forest*' in 1999, random forests are established according to the following procedure. First, a random sample is chosen from the total dataset for each tree. As the tree grows, a selection of a subset of the features takes place at each node. These serve as criteria for splitting the dataset. The target value is then determined for each *Decision Tree* individually. The averaging of these predictions represents the prediction of the Random Forest. [Bre01][Jam13]

The Random Forest has a number of hyperparameters. The most crucial one, besides the *maximum depth* of the trees `max_depth`, is the *number of decision trees* `n_estimators`. By default, the M*ean Square Error (MSE)* is used as criterion for splitting the dataset as the trees grow. [Sci18d]

```python
from sklearn.ensemble import RandomForestRegressor

# define used regressor
forest=RandomForestRegressor(n_estimators=20,
                             max_depth=10,
                             criterion='mse',
                            )

# train model
forest.fit(X_train,y_train)
```

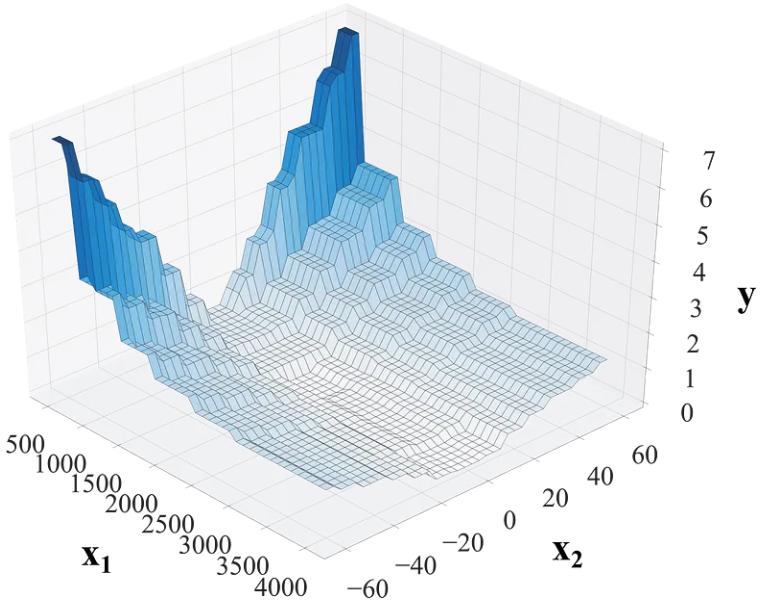
random_forest_regressor.py hosted with ❤️ by GitHub view raw

The following figure shows an example model for the Random Forest. The way it works results in the characteristic "*step*" form.

Model:

Random Forest

Number of Decision Trees: 20
Max. depth: 20



Random Forest: Sample Model — Image by the author

Since the Random Forest combines several models into a single one, it belongs to the field of ***Ensemble Learning***. More precisely, the Random Forest is a so-called *Bagging* technique.

Besides *Bagging*, the best known type of ensemble learning technique is *Boosting*, with the best known algorithms from this area being the *AdaBoost* and *XGboost* algorithms.

If you are interested in what exactly is the difference between boosting and bagging (and thus AdaBoost and Random Forest), you can find a more detailed introduction here:

Gaussian process regression

The *Gaussian Process* captures the typical behavior of a system on the basis of observations of a system and delivers as a result a probability distribution of possible interpolation functions for the problem at hand.

The *Gaussian Process Regression* makes use of the Bayes' theorem in the following, which is why it should be briefly explained in advance.

In general, the *Bayes' theorem* is defined as follows:

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)} \tag{10}$$

with: $P(A \mid B)$ = conditional probability of A occuring given that B is true (posterior)
$P(B \mid A)$ = conditional probability of B occuring given that A is true
$P(A)$ = independent probability of obsverving A (prior)
$P(B)$ = independent probability of obsverving B (prior)

It allows the inference from known values to unknown values. A often used application example is the disease detection. In the case of rapid tests, for example, one is interested in how high the actual probability is, that a positive tested person actual has the disease. [Fah16]

In the following, we will apply this principle to the *Gaussian Process*.

The *Gaussian Process* is defined by the expected value for each random variable, the `mean function m(x)` and a `covariance function k(x,x´)`.

$$(X) \sim \mathcal{GP}(m(x), k(x, x'))$$

The `mean function m(x)` reflects the *priori* function for the problem at hand and is based on known trends or biases in the data. If the expected value (mean function) is constant 0, it is called a centered Gaussian process.

$$m(x) := \mathbb{E}[f(x)]$$

The `covariance function k(x, x´)` , also called '*kernel*', describes the covariance of the random variables x and x' . These functions are analytically defined.

$$k(x, x') := \mathrm{Cov}(f(x), f(x')) = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))]$$

The kernel defines the shape and the course of the model functions and is used to describe, for example, abstract properties such as *smoothness, roughness* and *noise*. More kernels can be combined by certain computational rules to emulate systems with superimposed properties. [Ebd08][Kuß06] [Ras06][Vaf17]

In the following three of the most used kernels will be presented:

Squared Exponential Kernel

A popular *Kernel* is the `Squared Exponential Kernel` (Radial Basis Function) and has established itself as the '*standard kernel*' for the *Gaussian Process* and the *Support Vector Machine*. [Sci18l]

$$k_{\mathrm{SE}}(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right)$$

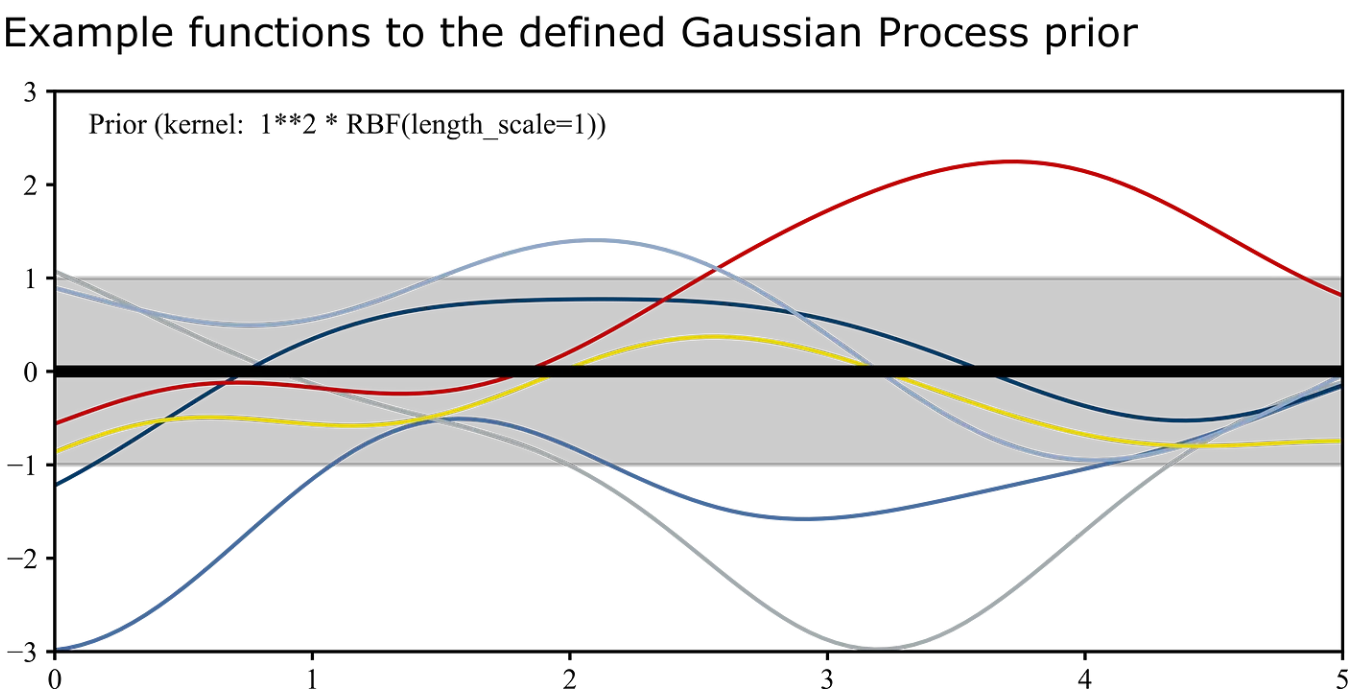
```python
from sklearn.gaussian_process.kernels import RBF
from sklearn.gaussian_process import GaussianProcessClassifier

# Definition of the Radial Basis Function (RBF) as kernel
kernel =1.0 * RBF( length_scale=1.0 , length_scale_bounds=(1e-1, 10.0))

# Train Gaussian Process
gpc = GaussianProcessClassifier(kernel=kernel, random_state=0).fit(X, y)
```

rbf_kernel.py hosted with ❤️ by GitHub view raw

The following figures show an example for an A-*priori*-Gaussian process `p(f)` through the `mean function m(x)` (black line) and the `confidence interval` (gray background). Generally, the confidence interval indicates the range, given an infinite repetition of a random experiment, with some probability, the true location of the parameter lies [Fah16][Enc18]. In this case, the boundaries of the confidence interval are defined by the *standard deviation σ*.

The colored curves represent some random functions of the *Gaussian Process*. The example curves serve only to abstract the form of the possible output functions. In principle, an infinite number of these curves could be created.

Squared Exponential Kernel



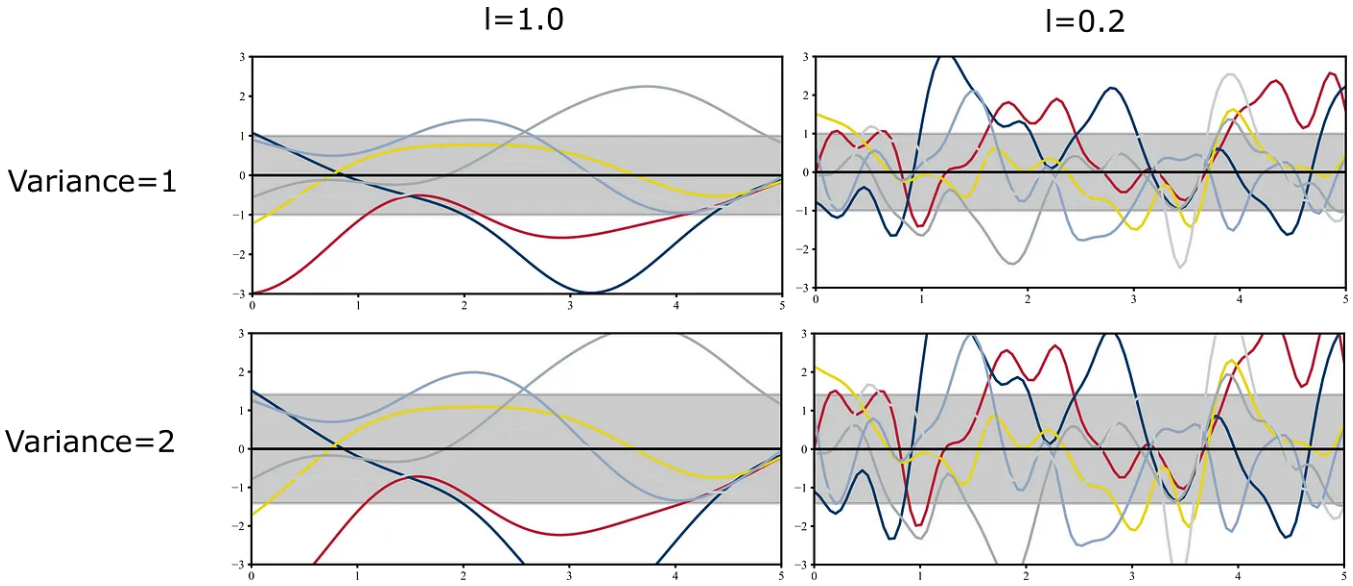
A-priori Gaussian-Prozess using a Squared Exponential Kernel — Image by the author(inspired by [Sci18n] [Duv14])

The kernel has only two hyperparameters:

- **l (length_scale)** decribes the characteristic length scale of the covariance function. The **length_scale** influences the length of the '*waves*' of the Gaussian functions.

- The **variance σ^2** defines the average distance of the function from its mean. The value should be chosen high for functions that cover a large range on the y-axis. [Ebd08]

The following figure shows the effects of the hyperparameters on the *A-Priori-Gaussian-Process* and its functions.



Squared Exponential Kernel: Influence of Hyperparameters — Image by the author

Rational Quadratic Kernel

The `Rational Quadratic Kernel` can be seen as a combination of several *Squared Exponential Kernels* with different `length_scale` settings (l). The parameter α determines the relative weighting of the '*large-scale*' and '*small-scale*' functions. When α approaches infinity, the Rational Quadratic Kernel is identical to the Squared Exponential Kernel. [Duv14][Sci18k][Mur12]

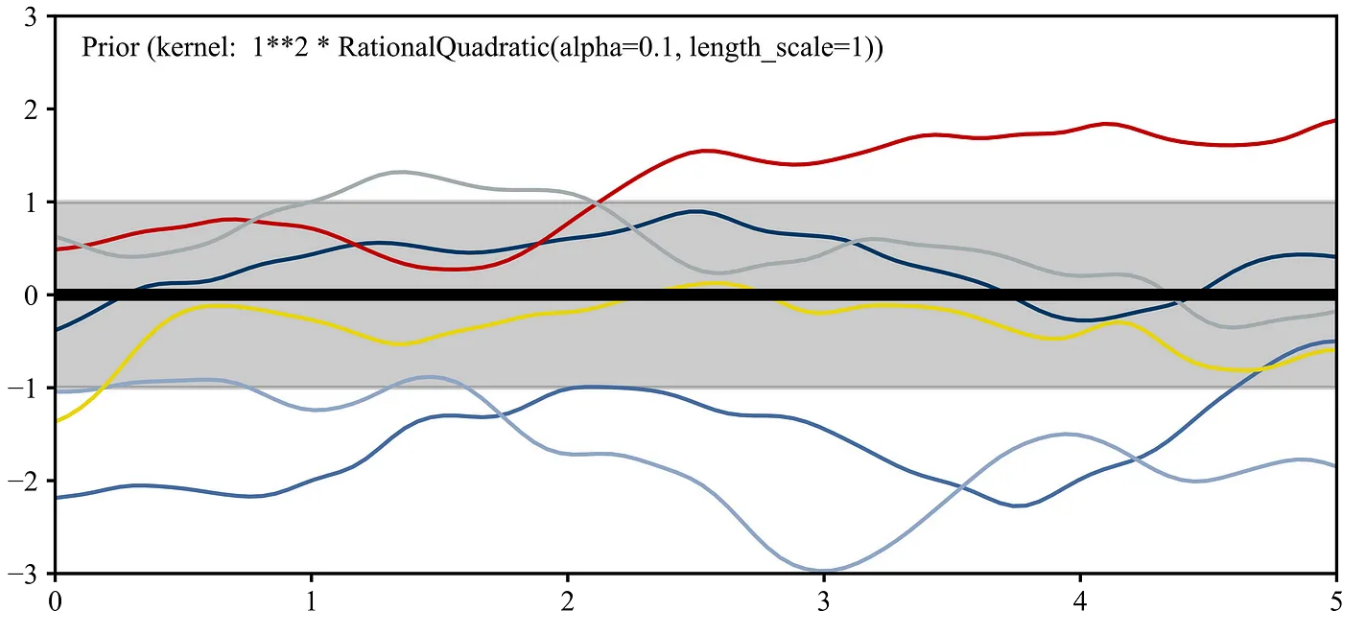
$$k_{\mathrm{RQ}}(x, x') = \sigma^2 \left(1 + \frac{(x - x')^2}{2\alpha\ell^2}\right)^{-\alpha}$$

```python
from sklearn.gaussian_process.kernels import RationalQuadratic
from sklearn.gaussian_process import GaussianProcessClassifier

# Definition of the Rational Quadratic Kernel
kernel = 1.0 * RationalQuadratic ( length_scale=1.0 , alpha=0.1)

# Train Gaussian Process
gpc = GaussianProcessClassifier(kernel=kernel, random_state=0).fit(X, y)
```

squared_exponential_kernel.py hosted with ❤️ by GitHub view raw

Rational Quadratic Kernel

Example functions to the defined Gaussian Process prior



A-priori Gaussian-Prozess using Rational Quadratic Kernel — Image by the author (inspired by [Sci18n] [Duv14])

Periodic Kernel

The `Periodic Kernel` allows functions to repeat themselves. The period p describes the distance between the repetitions of the function. The use of the '*lenghtscale*' parameter (l) is as mentioned earlier. [Sci18j]

$$k_{\mathrm{Per}}(x, x') = \sigma^2 \exp\left(-\frac{2\sin^2(\pi|x - x'|/p)}{\ell^2}\right)$$

A-priori Gaussian-Prozess using Periodic Kernel — Image by the author (inspired by [Sci18n][Duv14])

The `kernel funnction` and `mean function` together describe the *A-priori-Gaussian-Process*. With the help of some measured values, a ***A-posteriori-Gaussian-Process*** can be defined, which takes into account all available information about the problem. More precisely, no single solution results, but all possible functions of the interpolations, which are weighted with different probabilities. In the case of a regression task, specifically, the solution (function) with the highest probability is crucial. [Ras06][Wik18a][Wik18a]

For regression, typically a dataset with values of the independent variable $X \in R$ and associated values of the dependent variable $f \in R$ is given and one wants to predict output values f_* for new values X_*. [Vaf17]

For the simplest case, a process without noise, the multidimensional Gaussian distribution is defined as follows:

The covariance matrix can be divided into four parts. The covariance within the unknown values $K\_X_*X_*$, the covariance between the unknown and known K_X_*X values, and the covariance within the known values K_XX.

Since `f` is completely known, substituting the probability density into Bayes' theorem yields the *a-posterior-Gaussian-distribution*.

A detailed derivation is given by Rasmussen in his book '**Gaussian Processes for Machine Learning**'. [Ras06, p.8 ff.]

From priori to posteriori Gaussian Process: Explained with a simple example

In practice, a number of other kernels are used, including combinations of several kernel functions. The *constant kernel* for example is usually used in conjunction with others. Using this kernel without the combination with other kernels, usually makes no sense, because only constant correlations can be modeled. Nevertheless, in the following the constant kernel is used to explain and illustrate the Gaussian-process regression in a simple way.

The figure below shows the *a-priori-Gaussian-Process* with a variance of one. By defining the constant kernel as the covariance function, all sample functions show a parallel line to the x-axis.

Representation of the Gaussian process with the so-called constant kernel and one supporting data point — Image by the author

Since no statement is made in advance about a possible noise of the measured data, the process assumes that the given measurement point is a part of the true function. This limits the number of possible function equations to the straight line that passes directly through the point. Since the constant kernel only allows horizontal lines,

the number of possible lines in this simple case narrows down to exactly one possible function. Thus the covariance of the *a-posteriori-Gaussian process* is zero.

With the `RBF Kernel`, arbitrary processes can be mapped, but this time the result is not a single straight line as *A-posteriori-Gaussian,* but a multitude of functions. The function with the highest probability is the `mean function` of the *A-posteriori-Gaussian-Process*. The following figure shows the *A-posteriori-Gaussian-Process* and the used measurement points.

Posterior Gaussian Process with Squared Exponential Kernel and used data points — Image by the author

To implement the Gaussian Process in Python, the *A-priori-Gaussian-Process* must be defined in advance. The `mean function m(x)` is usually assumed to be constant and zero. By setting a parameter `normalize_y = True`, the process uses the mean of the dataset values as the constant expected value function. The choice of the covariance function is made by choosing the kernel. [Sci18m]

Gaussian Process Regression in Scikit-learn

The following source code describes how to implement the Gaussian Process Regression with scikit learn and the `RBF Kernel` used as covariance function. The first optimization process starts from the pre-set values (`length_scale` and `variance`) of the kernel. By the parameter `alpha` an assumption can be made in advance about the strength of the noise of the training data.

Optimization process: Hyperparameter erstimation using maximum-likelihood method

The hyperparameters are optimized during the fitting of the model by maximizing the *log-marginal likelihood (LML)*. *Maximum Likelihood Estimation (MLE)* is a method for determining the parameters of a statistical model. While the Regression methods already presented, such as Linear Regression, aim to minimize the *Mean Square Error*, the Gaussian-Process Regression tries to maximize the *likelihood function*. In other words, the parameters of the model are selected in such a way that the observed data appear most plausible according to their distribution.

In general, the `probability function f` of a `random variable X` is defined as:

This distribution is assumed to depend on a parameter ϑ. Given observed data, the probability can be considered as a function of ϑ :

Maximum likelihood estimators aim to maximize this function. Maxima are usually identified by differentiating the function and then setting it equal to zero. Since the *log likelihood function* has its maximum at the same point as the likelihood function, but is easier to calculate, it is usually used. [Goo16, p.128]

One speaks of a normal or Gaussian distribution if a random variable X has the following probability density [Fah16, p.83]:

The maximum-likelihood method will be explained in the following using a simple one-dimensional example. The figure below shows the dataset. All three plotted probability distributions reflect the distribution of the data. With maximum likelihood, one is interested in the distribution that is most likely.

The normal distribution depending on the parameters expected value and variance — Image by the author (inspired by [BB18])

The goal is to define the parameters σ^2 and μ in such a way that the probability is maximized over all data points considered. For example, given three data points with x-values 9, 10 and 13, similar to the data points in the figure, the joint probability is calculated from the individual probabilities as follows:

This function must be maximized. The mean value then corresponds to the `x-value` that is most likely to occur.

The following figure shows an example model for the Gaussian Process Regression.

Gaussian Process Regression: Sample Model — Image by the author

Support Vector Regression

The functionality of the Support Vector Regression (SVR) is based on the Support Vector Machine (SVM) and will first be explained with a simple example. We are looking for the linear function:

$$f(x) = \langle w, x \rangle + b$$

⟨w, x⟩ describes the cross product. The goal of SV Regression is to find a straight line as model for the data points whereas the parameters of the straight line should be defined in such a way that the line is as '*flat*' as possible. This can be achieved by minimizing the norm: [Wei18][Ber85]

For the model building process it does not matter how far the data points are from the modeled straight line as long as they are within a defined range (-ϵ to +ϵ). Deviations that exceed the specified limit ϵ are not allowed.

Functionality of the Support Vector Regression —Image by the author (inspired by [Smo04])

These conditions can be described as a convex optimization problem:

To this optimization problem finds a solution if all data points can be approximated within an accuracy of ϵ (The figure above shows a simple example for this scenario). However, this is a simplified assumption and usually does not hold in practice. To be able to circumvent unsolvable optimization problems, the variables ζi, $\zeta *$are introduced —the so called *slack variables*.

The soft margin loss setting for a linear SVM — Image by the author (inspired by [Smo04])

The figure above describes the "*punishment*"of deviations exceeding the amount of ϵ using a linear *loss function*. The loss function is called the *kernel*. Besides the linear kernel, the polynomial or RBF kernel are frequently in use. [Smo04][Yu12][Bur98]

Thus, the formulation according to Vapnik is as follows:

The *constant C* describes the trade-off between the condition of *flatness* and the deviations greater than ϵ that are tolerated.

In order to be able to model non-linear relationships with the Support Vector Regression as well, the so-called '*kernel trick*' is used. Therefore the original characteristics are mapped into a higher-dimensional space. [Pai12]

The implementation with `scikit-learn` and the *RBF kernel* looks like this:

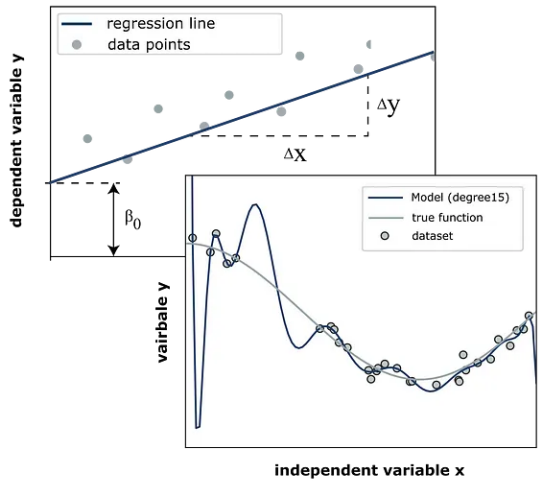
Summary of the regression methods presented in this article

The following figure provides an overview of the regression methods presented and a brief summary of how they work.

Linear and Polynomial Regression

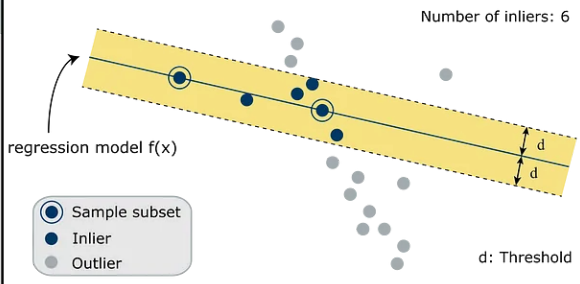
In Linear Regression, model building is done by minimizing the residual sum of squares (least squares method).

Polynomial regression is a combination of variable transformation and linear regression. Before the model building process, the input variables are transformed.



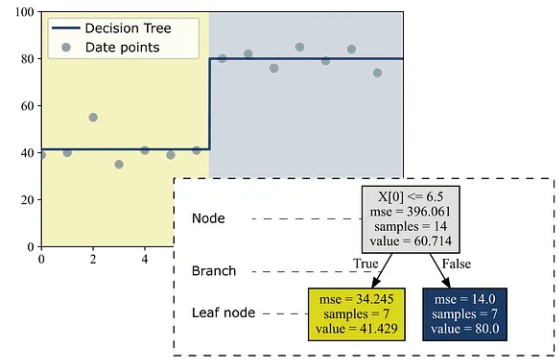
Robust Regression (RANSAC)

The RANSAC algorithm builds a linear model through a given sample and then counts the data points that are within a range away from this function. After iterative execution, the procedure selects the model with the highest number of inliers.



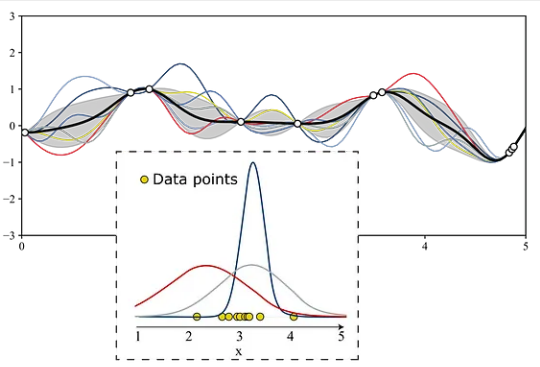
Decision Tree and Random Forest

The decision tree tries to split the dataset in such a way that the variance in the partial datasets is minimized. The Random Forest consists of a series of trees that take on different shapes as they grow through randomization. In forecasting, the Random Forest averages the results of the individual decision trees.



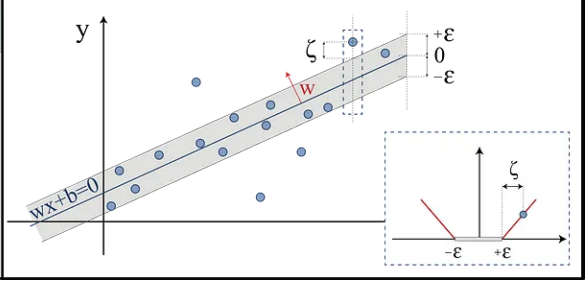
Gaussian Process Regression

The Gaussian process captures the typical behavior of a system based on observation and provides as a result a probability distribution of possible interpolation functions for the problem at hand. As a resulting model the process selects the most plausible function, i.e. the function with the highest probability.



Support Vector Regression

This method searches for a straight line that approximates the measurement points within a given accuracy. Since this exists in the rarest cases, larger deviations are evaluated with loss functions.



Overview of the presented regression methods — Image by the author

2. Model evaluation

There are various methods and procedures to evaluate the accuracy of a model.

Metric functions

The `sklearn.metrics` module includes several loss and evaluation functions to measure the quality of the regression models. *Mean Squared Error (MSE)* is a key criterion for assessing the quality of a regression model [Ras18, p.337]. If yˆ_i describes the value predicted by the model at the i-th data sample, and y_i describes the corresponding true value, then the ***Mean Squared Error (MSE)*** of the model over n_Samples is described as [Sci18a]:

Another parameter for determining the accuracy of regression models is the ***Mean Absolute Error (MAE)***.

Both metrics can be found in the module `sklearn.metrics` . They compare the predicted and actual values for the test dataset.

Coefficient of determination (R^2)

The so-called coefficient of determination (R^2) can be understood as a standardized version of the MSE. This allows an easier interpretation of the performance of the model. The best possible performance is described with the value 1.0. The R^2 -score can also become negative if the model shows arbitrary deviations from the truth value. A constant model, which makes the prediction of the values without the consideration of the input characteristics, would receive a R^2 -score of 0.0.

If y^_i describes the value predicted by the model at the i-th data sample, and y_i describes the associated true value, then the coefficient of determination R^2 over n_Samples is defined as [Sci18a]:

The output in Python is the function `r2_score`, where `y_true` is the true value of the dependent variable and `y_pred` is the value predicted by the model.

Cross validation in regression

Cross-validation is a statistical method for model selection. To evaluate a method, the entire dataset is divided into a training and a test dataset, whereby the training dataset usually comprises 80 to 90 % of the entire dataset. In order to achieve the best possible evaluation of the model, the aim is to have as **large a test dataset as possible**. Good model building is achieved by having as large a **training dataset as possible**.

Cross-validation is used to circumvent this dilemma. This method allows the entire dataset to be used for both training and testing. Compared to a fixed division into train and test data, *cross-validation* thus allows a more accurate estimate of model accuracy for future data or data not included in the dataset.

The k-fold cross validation divides the entire dataset `X` into `k` equal sized blocks (`X_1, …, X_k`). Then the algorithm is trained `k` times on `k-1` blocks and tested with the remaining block.

Functionality of cross-validation using the example of 5-fold cross-validation — Image by the author

Many learning methods allow an adjustment of the model complexity via one or more hyperparameters. This often leads to the problem of over- or underfitting. Cross-validation is used to find the optimal model complexity. The optimal complexity is achieved by minimizing the approximation error on a test dataset that is unknown during learning. [Du14, p. 27][Has09, p. 242]

The process already described is carried out for different parameter settings and model complexities. For the final model, the setting parameter (γ_opt) is chosen

that shows the lowest e*rror (e.g. MSE or MAE)*. For smaller training datasets, `k` can be equated to the number `n` of feature vectors. This method is called *Leave-One-Out Cross-validation*. [Ert16, p.233][Bow15, p.100]

The implementation with sklearn is done with the module `cross_validate` . The following code snippet shows the application of cross validation to evaluate the performance of the *Linear Regression.*

The `cv` value defines the number `k` of partitions into which the dataset is divided. The `Negativ Mean Squared Error` was used as the scoring parameter in this case. The squared error is passed to the list `scores` after each run. After the execution of the program code, `scores` represents a list with, in this case, three entries, i.e. the *Mean*

Square Error of each regression model. The models differ only in the choice of the test and training dataset, which are varied after each run as described earlier. [Sci18f][Coe13]

The function `cross_validate` uses the scoring parameters of the `.metrics` module. The following table describes a summary of the so-called scoring parameters used for the evaluation of regression models [Sci18e][Cod18].

Scoring parameters for the evaluation of regression models — Image by the author

Functions ending with `_score` return a value that should be maximized if possible. Functions ending with `_error` or `_loss` return a value to minimize.

If you take a look at the source code of the `sklearn.metrics.scorer` module, you can see that for all loss or error functions the parameter `greater_is_better` is set to FALSE and the scoring parameter are negated and supplemented with the expression `neg_` . This allows to handle all scoring parameters in the same way. [Cod18][Git18]

The following source code shows a typical application example of *cross-validation*. The example uses *Polynomial Regression* for modeling, which allows the setting of the model complexity via the specification of the *polynomial degree*. The image below shows the used dataset and the problem of overfitting. If the evaluation of the model were to be performed using the training data, models with higher complexity usually show higher accuracies. Since the dataset was generated using a sine function, the true function can be used for comparison. For this simple example, you can see at a glance that the polynomial regression model shown with polynomial degree 15 does not correctly represent the regression problem —it is an **overfitted model.**

Polynomial Regression: Overfitting — Image by the author

To be able to determine the “optimal” polynomial degree, *Cross-validation* is used in the following.

A good model is characterized by the lowest possible error of the model when applied to the test dataset. In order to obtain the best possible evaluation of the models for the relative small dataset, a ‘*leave-one-out cross-validation*’ is performed for different setting parameters. This is done by setting the number of partitions (into which the dataset is divided during cross-validation) to the number of data points. The list of `scores` obtained from cross-validation includes the *Mean Square Error* for each run. These values are averaged for an assessment of the used regression method.

The left graph of the following figure shows the results of cross-validation for different polynomial degrees. In addition, the error of the model from the training dataset is shown. The error decreases with increasing model complexity. This explains why an evaluation and optimization of the model on the basis of the training data is on the basis of the training data would not be practicable.

The *Mean Square Error* determined by cross-validation shows steadily low values in the range of a polynomial degree of three to seven. More complex systems no longer adequately represent the process, which is why the calculated accuracy for data not included in the training dataset and future data of the process decreases significantly. The optimum is shown at a polynomial degree of three. If the resulting model is plotted, it shows a good approximation to the 'true function'. (The 'true

function' of the dataset can be given in this case, because the data points were generated with a random offset to the given cos-function).

Choice of polynomial degree using cross-validation — Image by the author

3. Model building process

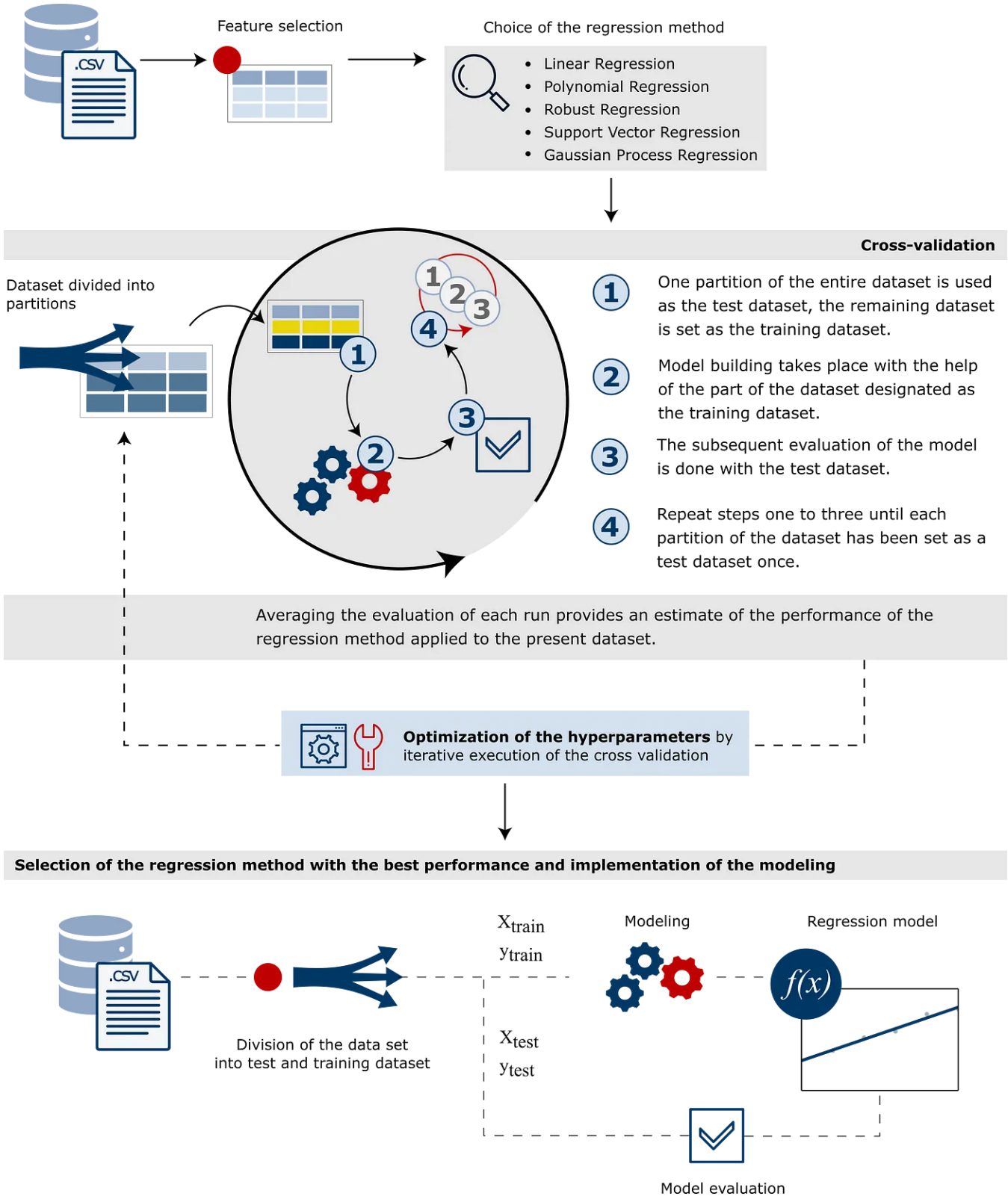
The figure below shows the schematic flow of the methode selection and subsequent model generation. The feature selection already takes place before the model building and defines the input attributes of the later regression model. The datasets were already structured during the creation in such a way that they only contain relevant attributes.

The regression methods are suitable for different problems, differently well. For evaluation, the dataset is split into training and test dataset before model building. This step is omitted in the source code, as this process is automatically performed iteratively during *cross-validation*. The execution of the cross validation is done by the `cross_val_score` function of the *scikit* library.

Cross-validation provides an indication of the performance of each regression method. For datasets with a small number of instances, a '*Leave One Out*' c*ross-validation* is usually performed. For this, the partition number of the *cross-validation* is set equal to the length of the dataset.

Hyperparameter optimization through repetitive cross-validation with different hyperparameter settings:

The result of the *cross-validation* represents a list with the values of the selected scoring parameters. Since the evaluation is performed after each run, if the dataset is divided into five partitions, there is also a list with five evaluation values. An averaging of these values allows an assessment of the performance of the regression procedure. Since most regression methods allow an adjustment of the model complexity via one or more hyperparameters, an adjustment of the hyperparameters is necessary for a meaningful comparison of the regression methods. The finding of these optimal hyperparameter settings is done by iterative model building. The *cross-validation* is performed repeatedly for different hyperparameter settings. Finally, the parameter settings are chosen which showed the best model accuracy during the evaluation. This process is performed by loops which automatically change the hyperparameters within certain limits and store the evaluation values. The selection of the optimal settings is then done by manual or automated search for the best evaluation results.



Schematic representation of the evaluation of the different regression methods and subsequent model building — Image by the author

While Linear Regression does not allow setting the model complexity, most algorithms comprise multiple hyperparameters. For the optimization of the model, it is usually not sufficient to vary only one of the hyperparameters in procedures with several hyperparameter setting options. Care must be taken that the hyperparameters are not exclusively considered individually, since the effects of the parameter changes partly influence each other.

The figure below shows a list of some important hyperparameters of the presented methods. Especially for the methods which use kernel functions to find the solution, the number of possible settings goes far beyond the listed ones. For a more detailed description, you can find a comprehensive documentation to the methods and their hyperparameters at: scikit-learn.org.

Overview of the most important hyperparameters of the regression methods — Image by the author

You can find a more detailed introduction into the field of hyperparameter optimization and used methodologies like Grid Search or Bayesian Optimization here:

A Step-by-Step Introduction to Hyperparameter Tuning, Grid Search and Bayesian Optimization

An illustrative explanation of Bayesian optimization using regression problems

towardsdatascience.com

Summary

Hope I could give you an overview of different techniques used for regression analysis. Of course, the article does not claim to give a complete picture of regression. Neither with regard to the field of regression nor to the concepts presented. Many important algorithms were not mentioned at all. Nevertheless, these 7 algorithm give you a first good overview of techniques that are used and how they differ from each other in the way they work.

If you found the article helpful, you can also find a similar article on concepts and algorithms used for *Anomaly Detection*:

A Comprehensive Beginners Guide to the Diverse Field of Anomaly Detection

Isolation Forest, Local Outlier Factor, One-Class SVM, Autoencoders, Robust Covariance Estimator and Time Series...

towardsdatascience.com

If you are not yet a Medium Premium member and want to become one, you can support me by signing up using this referral link.

Thank you for reading!!

References

[Aun18] Aunkofer, B. Entscheidungsbaum-Algorithmus ID3 — Data Science Blog, 2018. URL https://data-science-blog.com/blog/2017/08/13/entscheidungsbaum-algorithmus-id3/

[BB18] Brooks-Bartlett, J. Probability concepts explained: Maximum likelihood estimation, 2018. URL https://towardsdatascience.com/probability/concepts-explained-maximum-likelihood-estimation-c7b4342fdbb134

[Bel15] Bell, J. Machine learning: Hands-on for developers and technical professionals. Wiley, Indianapolis Indiana, 2015. ISBN 1118889061

[Ber85] Berger, J. O. Statistical Decision Theory and Bayesian Analysis. Springer Series in Statistics. Springer, New York, NY, second edition Auflage, 1985. ISBN 9781441930743. doi:10.1007/978–1–4757–4286–2. URL http://dx.doi.org/10.1007/978-1-4757-4286-2

[Bow15] Bowles, M. Machine Learning in Python: Essential techniques for predictive analysis. John Wiley & Sons Inc, Indianapolis IN, 2015. ISBN 1118961749B

[Bre01] Breiman, L. Random Forest. 2001.

[Bur98] Burges, C. J. C.; Kaufman, L.; Smola, A. J.; Vapnik, V. Support Vector Regression Machines. 1998. URL http://papers.nips.cc/paper/1238-support-vector-regression-machines.pdfine

[Cal03] Callan, R. Neuronale Netze im Klartext. Im Klartext. Pearson Studium, Műnchen and Harlow, 2003. ISBN 9783827370716

[Cod18] CodeExamples. 3.3. Modellbewertung: Quantifizierung der Quali[1]tät von Vorhersagen | scikit-learn Documentation | CODE Examples

[Coe13] Coelho, L. P.; Richert, W. Building Machine Learning Systems with Python. 2013

[Dan18] Daniilidis, K. RANSAC: Random Sample Consensus I — Pose Estimation | Coursera, 2018. URL https://www.coursera.org/lecture/robotics-perception/ransac-random-sample-consensus-i-z0GWq

[Du14] Du, K.-L.; Swamy, M. N. S. Neural Networks and Statistical Learning. Springer London, London, 2014. ISBN 978–1–4471–5570–6. doi:10.1007/978–1–4471-5571–3

[Duv14] Duvenaud, D. K. Automatic Model Construction with Gaussian Processes. 2014. URL https://www.cs.toronto.edu/~duvenaud/thesis.pdfin

[Ebd08] Ebden, M. Gaussian Processes for Regression: A Quick Introduction. 2008.

[Enc18] The significance test controversy and the bayesian alternative, 21.06.2018. URL https://www.encyclopediaofmath.org/index.php/The_significance_test_controversy_and_the_bayesian_alternative

[Ert16] Ertel, W. Grundkurs Künstliche Intelligenz: Eine praxisorientierte Einführung. Computational Intelligence. Springer Fachmedien Wiesbaden GmbH and Springer Vieweg, Wiesbaden, 4., überarb. aufl. 2017 Auflage, 2016. ISBN 9783658135485

[Fah16] Fahrmeir, L.; Heumann, C.; Künstler, R. Statistik: Der Weg zur Datenanalyse. Springer-Lehrbuch. Springer Spektrum, Berlin and Heidelberg, 8., überarbeitete und ergänzte auflage Auflage, 2016. ISBN 978–3–662 50371–3. doi:10.1007/978–3–662–50372–0

[Fis80] Fischler, M.; Bolles, R. Random Sample Consensus: A paradigm for model fitting with applications to image analysis and automated cartography. 1980. URL http://www.dtic.mil/dtic/tr/fulltext/u2/a460585.pdf0585.p

[Git18] GitHub. sklearn.metrics-Quellcode, 2018.

[Goo16] Goodfellow, I.; Bengio, Y.; Courville, A. Deep learning. MIT Press, Cambridge, Massachusetts and London, England, 2016. ISBN 9780262035613. URL http://www.deeplearningbook.org/

[Has09] Hastie, T.; Tibshirani, R.; Friedman, J. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer New York, New York, NY, 2009. ISBN 978–0–387–84857–0. doi:10.1007/b94608

[Hub05] Huber, P. J. Robust statistics. Wiley, New York, NY, 2005. ISBN 0–47141805-6

[Kuß06] Kuß, M. Gaussian Process Models for Robust Regression, Classification, and Reinforcement Learning. 2006. URL http://tuprints.ulb.tu-darmstadt.dde/epda/000674/GaussianProcessModelsKuss.pdfKu

[Mur12] Murphy, K. P. Machine learning: A probabilistic perspective. Adaptive computation and machine learning series. MIT Press, Cambridge, Mass., 2012. ISBN 9780262018029. URL https://ebookcentral.proquest.com/auth/lib/subhh/login.action?returnURL=https%3A%2F%2Febookcentral.proquest.com%2Flib%2Fsubhh%2Fdetail.action%3FdocID%3D3339490

[Pai12] Paisitkriangkrai, P. Linear Regression and Support Vector Regression. 2012.URL https://cs.adelaide.edu.au/~chhshen/teaching/ML_SVR.pdf

[Qui86] Quinlan, J. R. Induction of decision trees. Machine Learning, 1(1):81–106, 1986. ISSN 0885–6125. doi:10.1007/BF00116251. URL https://link.springer.com/content/pdf/10.1007%2FBF00116251.pdf

[Ras06] Rasmussen, C. E.; Williams, C. K. I. Gaussian Processes for Machine Learning. 2006

[Ras18] Raschka, S.; Mirjalili, V. Machine Learning mit Python und Scikit-Learn und TensorFlow: Das umfassende Praxis-Handbuch für Data Science, Deep Learning und Predictive Analytics. mitp, Frechen, 2., aktualisierte und erweiterte auflage Auflage, 2018. ISBN 9783958457331

[Rod04] Rodehorst, V. Photogrammetrische 3D-Rekonstruktion im Nahbereich durch Auto-Kalibrierung mit projektiver Geometrie: Zugl.: Berlin, Techn. Univ., Diss., 2004. wvb Wiss. Verl. Berlin, Berlin, 2004. ISBN 978–3–936846–83–6

[Sci18a] ScikitLearn. 3.3. Model evaluation: quantifying the quality of predictions — scikit-learn 0.20.0 documentation, 05.10.2018. URL https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics

[Sci18c] ScikitLearn. sklearn.tree.DecisionTreeRegressor — scikit-learn 0.20.0 documentation, 08.11.2018. URL https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

[Sci18e] ScikitLearn. 3.3. Model evaluation: quantifying the quality of predictions — scikit-learn 0.20.0 documentation, 24.10.2018. URL https://scikit-learn.org/stable/modules/model_evaluation.html

[Sci18f] ScikitLearn. sklearn.model_selection.cross_val_score — scikitlearn 0.20.0 documentation, 24.10.2018. URL https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html#sklearn.model_selection.cross_val_score

[Sci18g] ScikitLearn. 4.1. Pipelines and composite estimators — scikit-learn 0.20.0 documentation, 26.10.2018. URL https://scikit-learn.org/stable/modules/compose.html

[Sci18h] ScikitLearn. sklearn.preprocessing.PolynomialFeatures — scikit-learn 0.20.0 documentation, 26.10.2018. URL https://scikit-

learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html#sklearn.preprocessing.PolynomialFeatures

[Sci18j] ScikitLearn. sklearn.gaussian_process.kernels.ExpSineSquared — scikit-learn 0.20.0 documentation, 27.10.2018. URL https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.kernels.ExpSineSquared.html#sklearn.gaussian_process.kernels.ExpSineSquared

[Sci18k] ScikitLearn. sklearn.gaussian_process.kernels.RationalQuadratic — scikit-learn 0.20.0 documentation, 27.10.2018. URL https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.kernels.RationalQuadratic.html#sklearn.gaussian_process.kernels.RationalQuadratic

[Sci18m] ScikitLearn. 1.7. Gaussian Processes — scikit-learn 0.20.1 documentation, 28.11.2018. URL https://scikit-learn.org/stable/modules/gaussian_process.html

[Sci18n] ScikitLearn. Illustration of prior and posterior Gaussian process for different kernels — scikit-learn 0.20.0 documentation, 31.10.2018. URL https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpr_prior_posterior.html#sphx-glr-auto-examples-gaussian-process-plot-gpr-prior-posterior-py

[Smo04] Smola, A. J.; Schölkopf, B. A tutorial on support vector regression. Statistics and Computing, 14(3):199–222, 2004. ISSN 0960–3174. doi:10.1023/B:STCO.0000035301.49549.8849549.

[Vaf17] Vafa, K. Gaussian Process Tutorial, 2017. URL http://keyonvafa.com/gp-tutorial/

[Wei18] Weisstein, E. L2-Norm, 2018. URL http://mathworld.wolfram.com/L2-Norm.html

[Wie12] Wieland, A.; Marcus Wallenburg, C. Dealing with supply chain risks. International Journal of Physical Distribution & Logistics Management, 42(10):887–905, 2012. ISSN 0960–0035. doi:10.1108/09600031211281411. URL https://www.emeraldinsight.com/doi/pdfplus/10.1108/09600031211281411

[Wik18a] Gauß-Prozess, 13.10.2018. URL https://de.wikipedia.org/w/index.php?oldid=181728459

[Yu12] Yu, H.; Kim, S. SVM Tutorial — Classification, Regression and Ranking. G. Rozenberg; T. Bäck; J. N. Kok, Handbook of Natural Computing, 479–506. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978–3–540–92909–3.

Linear Regression | Support Vector Regression | Random Forest Regressor | Editors Pick | Getting Started

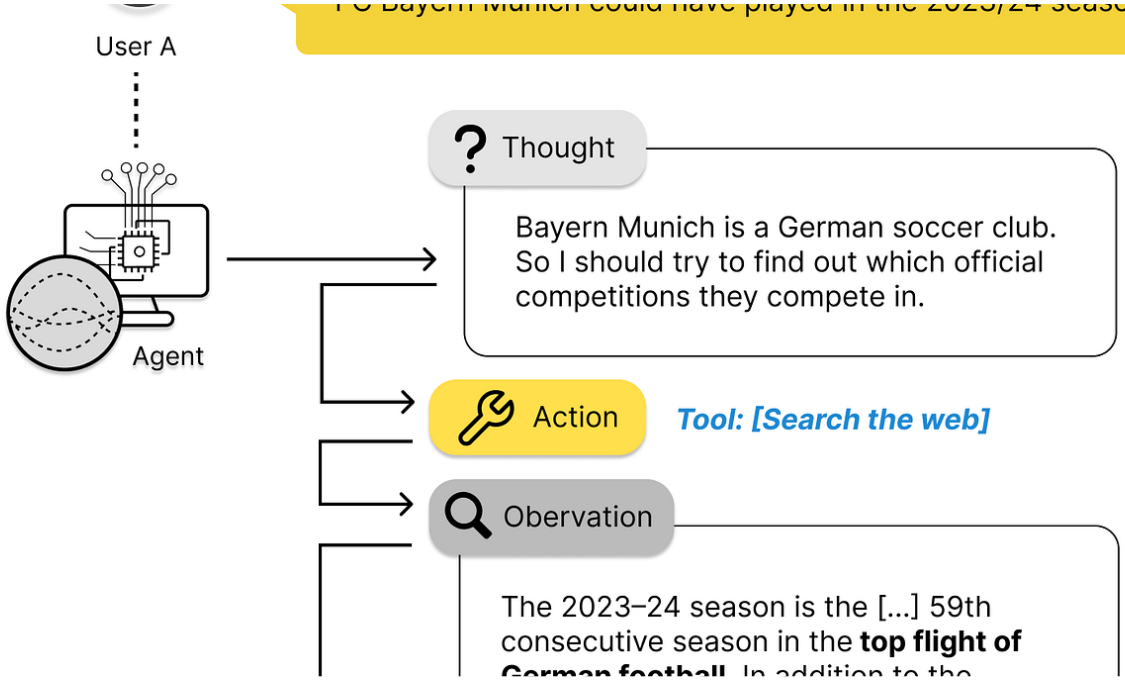
Follow

Written by Dominik Polzer

2.4K Followers · Writer for Towards Data Science

Machine Learning Engineer. I like to break down machine learning concepts into easy to understand pieces. linkedin.com/in/polzerdo/

More from Dominik Polzer and Towards Data Science



Dominik Polzer in Towards Data Science

Navigating the World of LLM Agents: A Beginner's Guide

A Step-by-Step Guide to Discover and Harness the Power of LLM Agents and Toolkits

· 19 min read · Jan 10, 2024

745 7

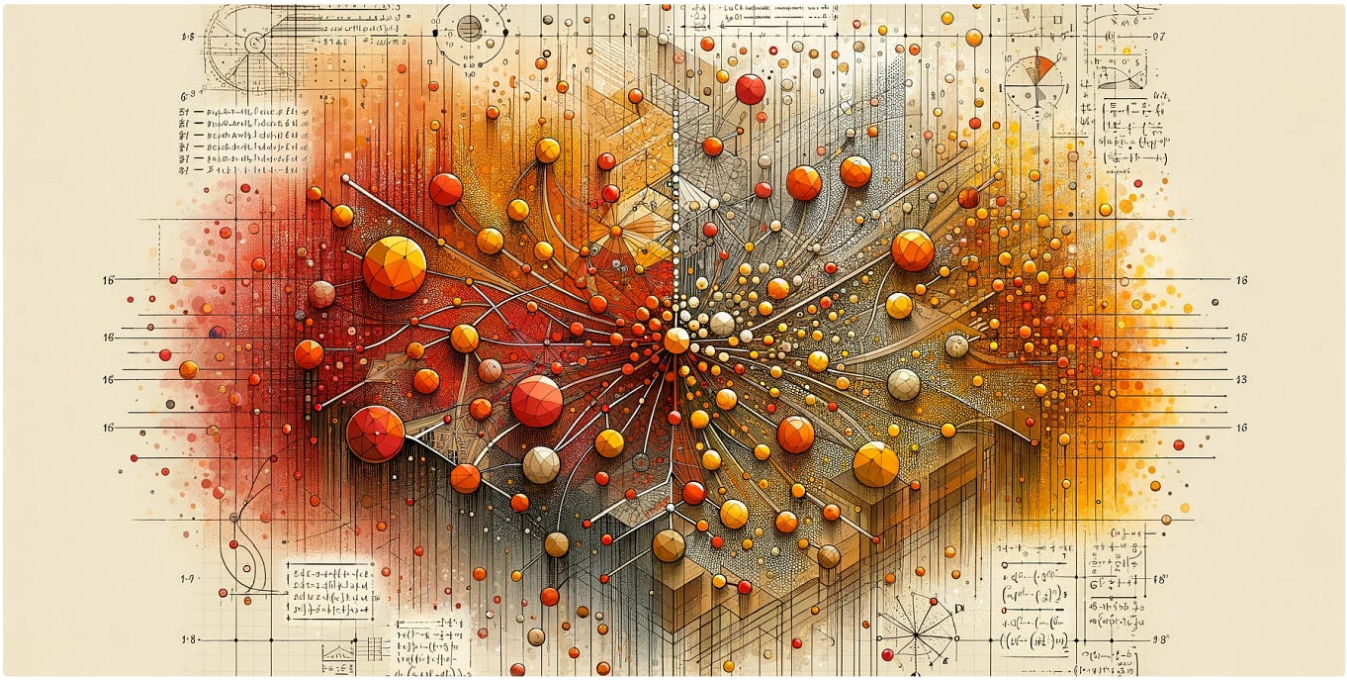
Cristian Leo in Towards Data Science

The Math behind Adam Optimizer

Why is Adam the most popular optimizer in Deep Learning? Let's understand it by diving into its math, and recreating the algorithm.

16 min read · Jan 30, 2024

1.94K 14



Cristian Leo in Towards Data Science

The Math Behind K-Nearest Neighbors

Why is KNN one of the most popular machine learning algorithm? Let's understand it by diving into its math, and building it from scratch.

18 min read · Feb 6, 2024

1K 5

Dominik Polzer in Towards Data Science

All You Need to Know to Build Your First LLM App

A step-by-step tutorial to document loaders, embeddings, vector stores and prompt templates

✨ · 26 min read · Jun 22, 2023

5.2K 48

See all from Dominik Polzer

See all from Towards Data Science

Recommended from Medium

Cristian Leo in Towards Data Science

The Math behind Adam Optimizer

Why is Adam the most popular optimizer in Deep Learning? Let's understand it by diving into its math, and recreating the algorithm.

16 min read · Jan 30, 2024

1.94K 14

James Presbitero Jr. in Practice in Public

These Words Make it Obvious That Your Text is Written By AI

These 7 words are painfully obvious. They make me cringe. They will make your reader cringe.

5 min read · Dec 31, 2023

39K 1040

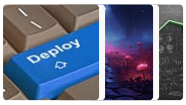
Lists

Practical Guides to Machine Learning

10 stories · 1086 saves

The New Chatbots: ChatGPT, Bard, and Beyond

12 stories · 309 saves



Predictive Modeling w/ Python

20 stories · 923 saves

Natural Language Processing

1219 stories · 692 saves

Artturi Jalli

I Built an App in 6 Hours that Makes $1,500/Mo

Copy my strategy!

· 3 min read · Jan 23, 2024

10K 129

Anmol Tomar in CodeX

Say Goodbye to Loops in Python, and Welcome Vectorization!

Use Vectorization — a super-fast alternative to loops in Python

· 5 min read · Dec 28, 2023

4.3K 50

Austin Starks in Artificial Intelligence in Plain English

Reinforcement Learning is Dead. Long Live the Transformer!

Large Language Models are more powerful than you imagine

8 min read · Jan 13, 2024

1.1K 33

Nikhil Adithyan in DataDrivenInvestor

Stock Price Prediction with Quantum Machine Learning in Python

An overview of the challenges and opportunities

17 min read · Jan 23, 2024

1.6K 17