UNIVERSITY OF MANNHEIM

## Data Mining

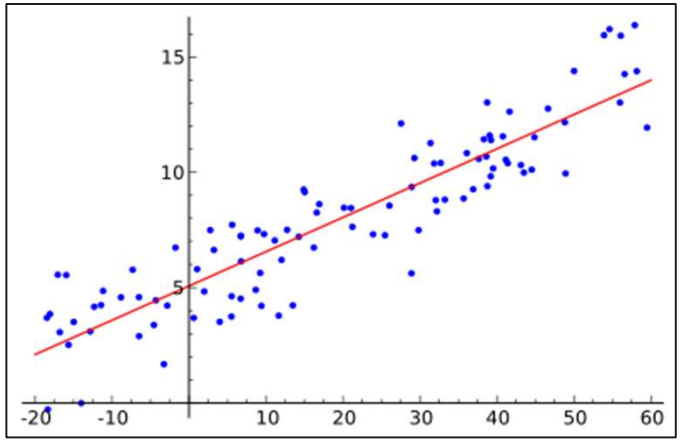
# Regression

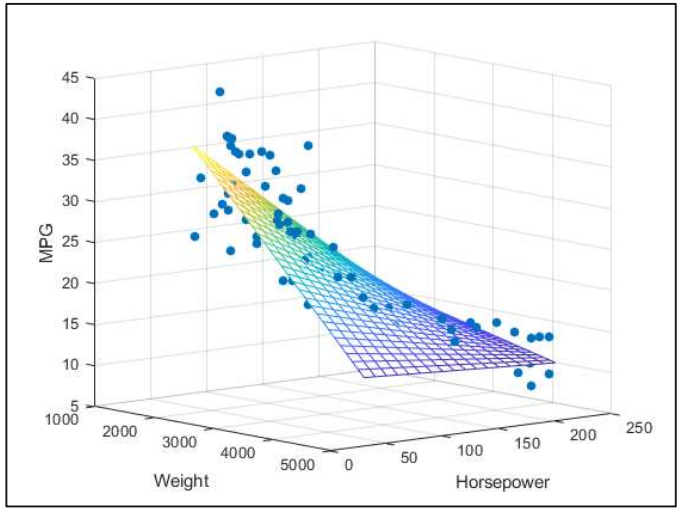
# Outline

1. What is Regression?
2. KNN for Regression
3. Model Evaluation
4. Regression Trees
5. Linear Regression
6. Polynominal Regression
7. Local Regression
8. ANNs for Regression
9. Time Series Forecasting
10. The Bias/Variance-Tradeoff

# 1. What is Regression?

- Goal: Predict the value of a continuous variable based on the values of other variables assuming a linear or nonlinear model of dependency
  - The predicted variable is called dependent and is denoted $\hat{y}$
  - The other variables are called explanatory variables or independent variables denoted $X = x_1, x_2, \ldots, x_n$
- Approach: Given training examples $(X_i, y_i)$ learn a model $f$ to predict $\hat{y}$ from $X_{unseen}$
- Difference to classification: The predicted attribute is continuous, while classification is used to predict nominal class attributes



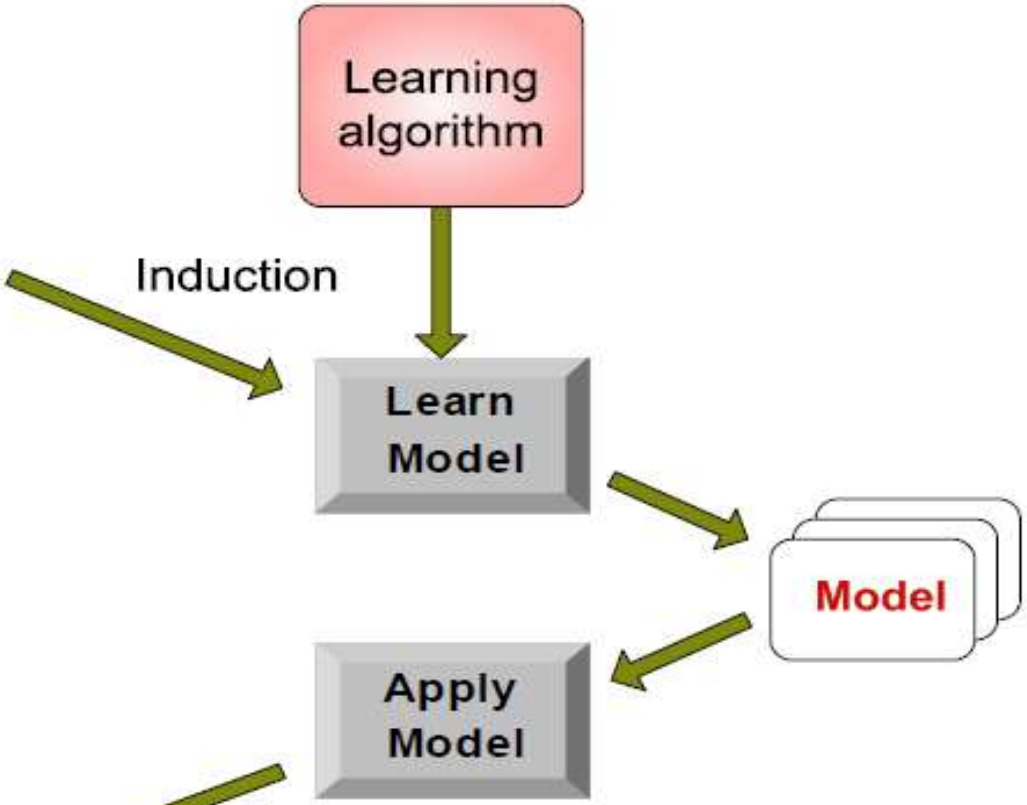
# Regression Model Learning and Application

Explanatory variables X

Dependent variable y

| Tid | Attrib1 | Attrib2 | Attrib3 | y |
|---|---|---|---|---|
| 1 | Yes | Large | 125K | 12 |
| 2 | No | Medium | 100K | 194 |
| 3 | No | Small | 70K | 4.5 |
| 4 | Yes | Medium | 120K | 78 |
| 5 | No | Large | 95K | 2.4 |
| 6 | No | Medium | 60K | 89 |
| 7 | Yes | Large | 220K | 3.1 |
| 8 | No | Small | 85K | 244 |
| 9 | No | Medium | 75K | 669 |
| 10 | No | Small | 90K | 4 |

Training Set

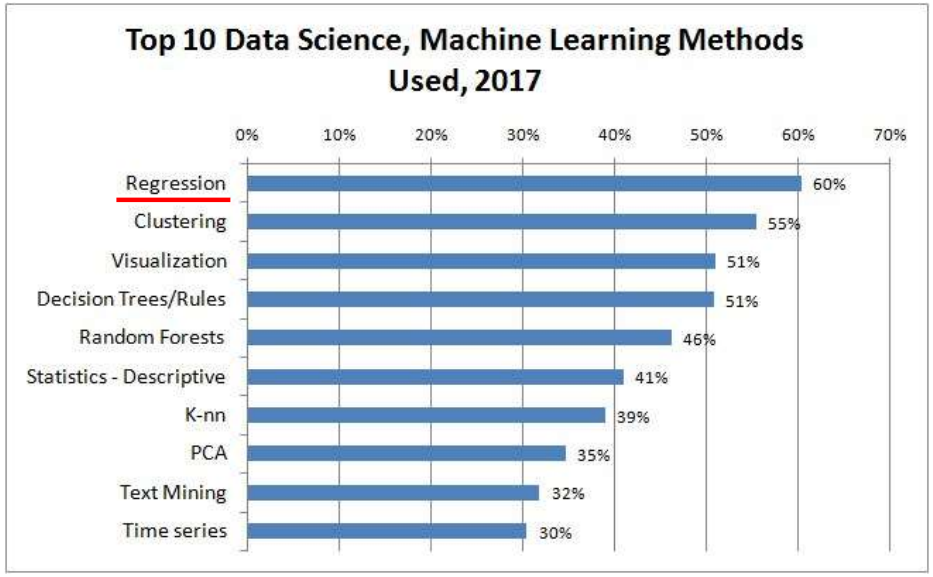



| Tid | Attrib1 | Attrib2 | Attrib3 | $\hat{y}$ |
|---|---|---|---|---|
| 11 | No | Small | 55K | ? |
| 12 | Yes | Medium | 80K | ? |
| 13 | Yes | Large | 110K | ? |
| 14 | No | Small | 95K | ? |
| 15 | No | Large | 67K | ? |

Unseen Records

# Application Examples

- House Market
  - dependent: price of a house
  - explanatory variables: rooms, distance to public transport, size of garden
- Gasoline Consumption
  - dependent: MPG (miles per gallon)
  - explanatory variables: weight of car, horsepower, type of engine
- Weather Forecasting
  - dependent: wind speed
  - explanatory variables: temperature, humidity, air pressure change
- Stock Market
  - dependent: price of a share
  - explanatory variables: company profit, sector outlook, month of year, mood of investors




Source: KDnuggets online poll, 732 votes

# Regression Techniques

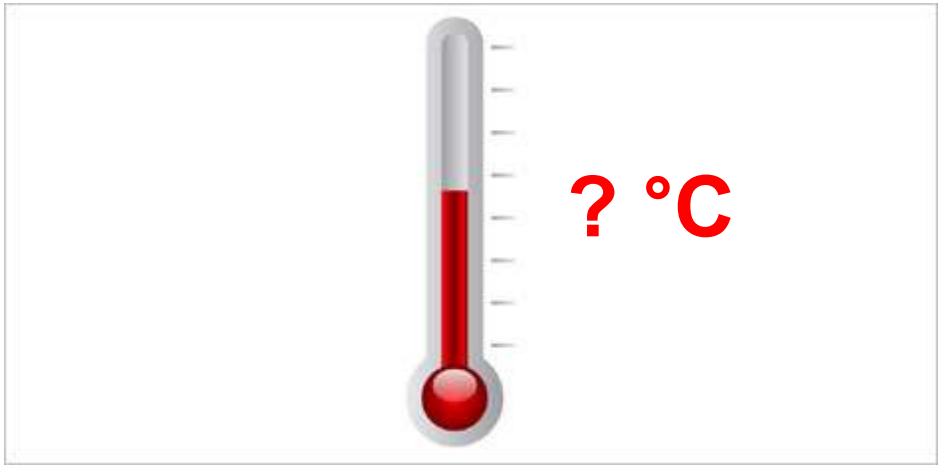
1. Linear Regression
2. Polynomial Regression
3. Local Regression
4. K-Nearest-Neighbors Regression
5. Regression Trees
6. Artificial Neural Networks
7. Deep Neural Networks
8. Component Models of Time Series

…

# 2. K-Nearest-Neighbors Regression

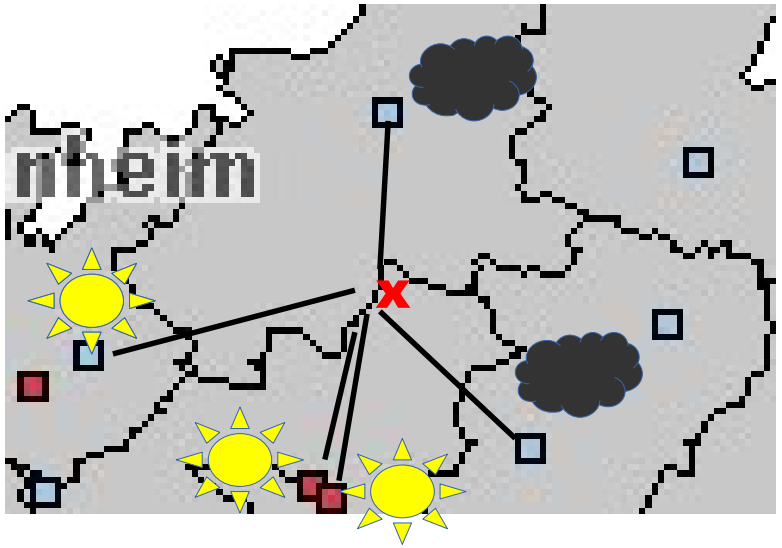
## Problem

- predict the temperature in a certain place
- where there is no weather station
- how could you do that?




# Recap: K-Nearest-Neighbors Classification

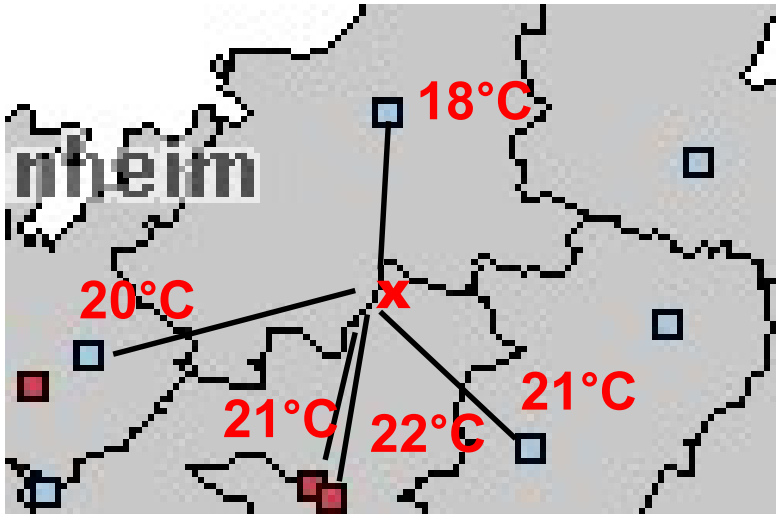
- Idea: Vote of the nearest stations
- Example:
  - 3x sunny
  - 2x cloudy
  - Result: sunny




- Approach is called
  - “k nearest neighbors”
  - where k is the number of neighbors to consider
  - in the example: k=5
  - in the example: “near” denotes geographical proximity

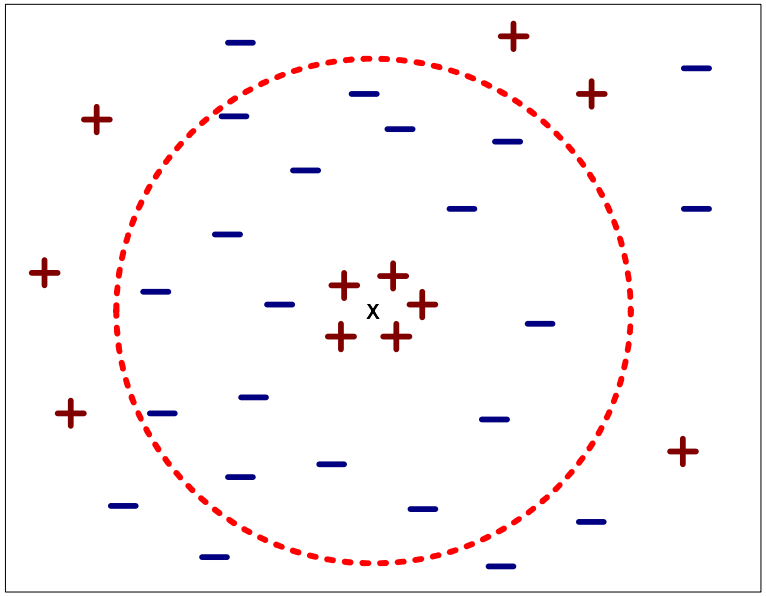
# K-Nearest-Neighbors Regression

- Idea: use the numeric average of the nearest stations
- Example:
  - 18°C, 20°C, 21°C, 22°C, 21°C
- Compute the average
  - again: k=5
  - average = (18+20+21+22+21) / 5
  - prediction: $\hat{y}$ = 20.4°C



# Choosing a Good Value for K

- All considerations from KNN classification also apply to KNN regression
  - If k is too small, the result is sensitive to noise points
  - If k is too large, local patterns may be averaged out
- Rule of thumb: Test k values between 1 and 20



# K-Nearest-Neighbor Regression in Python and RapidMiner

RapidMiner


| Name | Type | Missing | Statistics | | |
|---|---|---|---|---|---|
| Filter (4 / 4 attributes): Search for Attributes | | | | | |
| Label **label** | Real | 0 | Open visualizations | Min -970.881 | Max 974.564 | Average 5.553 |
| **att1** | Real | 0 | Min -9.968 | Max 9.777 | Average -0.591 |
| **att2** | Real | 0 | Min -9.920 | Max 9.973 | Average 0.377 |
| | | | | | Average 0.328 |

Python

```python
from sklearn.neighbors import KNeighborsRegressor

# Create and fit a KNN regressor
estimator = KNeighborsRegressor(n_neighbors=15)
estimator.fit(training_set_X, training_dependent_y)

# Make predictions for unseen examples
y_hat = estimator.predict(test_set_X)
```

# Numeric Predictions are Added to the Dataset

Prediction

ExampleSet (Cross Validation)

Data | Statistics | Charts | Advanced Charts

ExampleSet (200 examples, 2 special attributes, 5 regular attributes)  Filter (200 / 200 examples): all

| Row No. | label | prediction(label) | a1 | a2 | a3 | a4 | a5 |
|---|---|---|---|---|---|---|---|
| 1 | 138.755 | 129.424 | 4.595 | 4.388 | 4.926 | 2.682 | 8.618 |
| 2 | 56.200 | 52.996 | 2.667 | 4.173 | 2.484 | 2.867 | 1.586 |
| 3 | 39.974 | 40.865 | 2.654 | 2.904 | 3.093 | 3.880 | 7.181 |
| 4 | 428.422 | 399.541 | 5.330 | 8.735 | 6.563 | 4.917 | 4.344 |
| 5 | 80.938 | 66.471 | 4.423 | 1.715 | 9.285 | 2.903 | 2.749 |
| 6 | | | | | | | |
| 7 | | | | | | | |
| 8 | | | | | | | |
| 9 | | | | | | | |
| 10 | | | | | | | |
| 11 | | | | | | | |

ExampleSet (Cross Validation)

Data | Statistics | Charts

Filter (7 / 7 attributes): Search for Attributes

| Name | Type | Missing | Statistics | | |
|---|---|---|---|---|---|
| Label **label** | Real | 0 | Min 0.143 | Max 926.739 | Average 180.418 |
| Prediction **prediction(label)** | Real | 0 | Min -32.614 | Max 898.428 | Average 177.064 |
| **a1** | Real | 0 | Min 0.081 | Max 9.940 | Average 5.000 |
| **a2** | Real | 0 | Min 0.009 | Max 9.986 | Average 4.812 |

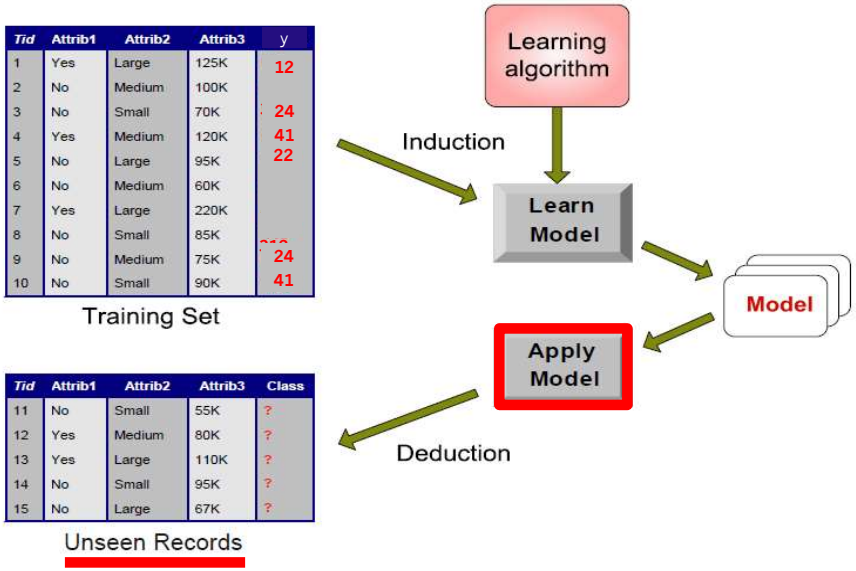
Prediction

# 3. Model Evaluation

Central Question:

How good is a model at predicting the dependent variable for unseen records?

(generalization performance)




## 3.1 Methods for Model Evaluation

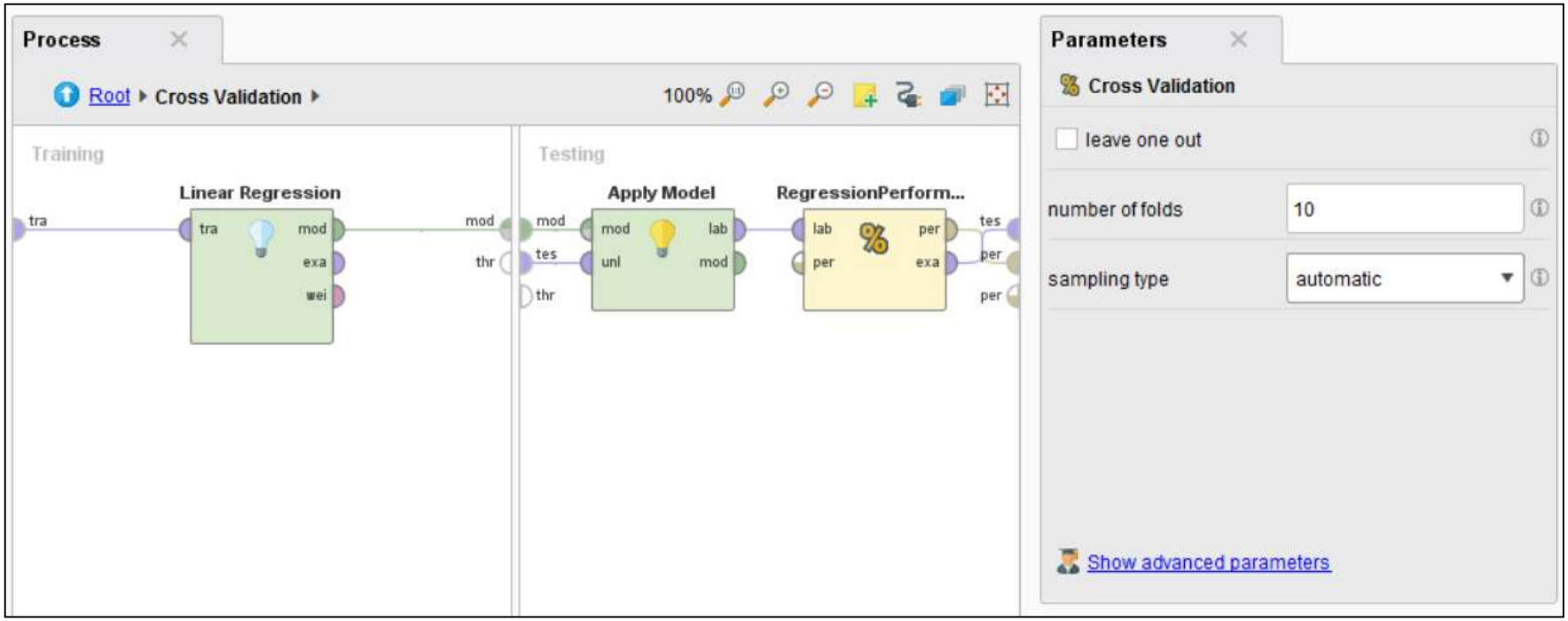
- How to obtain reliable estimates?

## 3.2 Metrics for Model Evaluation

- How to measure the performance of a regression model?

# 3.1 Methods for Model Evaluation

- The same considerations apply as for classification
  - Cross Validation: 10-fold (90% for training, 10% for testing in each iteration)
  - Holdout Validation: 80% random share for training, 20% for testing
- Estimating performance metrics in RapidMiner
  - Cross Validation Operator + Regression Performance Operator



# Nested Cross-Validation for Hyperparameter Selection

- Uses inner cross validation to select best hyperparameter values
- Uses outer cross validation to estimate generalization error of models learned using best hyperparameter values

Python

```python
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsRegressor

# Create KNN regressor
estimator_knn = KNeighborsRegressor()

# Specify the hyperparameter values for the search
grid = {"n_neighbors": range(1,20)}

# Create the grid search estimator for model selection
estimator_gs = GridSearchCV(estimator_knn, grid, cv=5, scoring='neg_mean_squared_error')

# Run nested cross-validation for model evaluation
mse_cv = cross_val_score(estimator_gs, X, y, cv=5, scoring='neg_mean_squared_error')
```

# 3.2 Metrics for Model Evaluation

- Mean Absolute Error (*MAE*) computes the average deviation between predicted value $p_i$ and the actual value $r_i$

$$MAE = \frac{1}{n}\sum_{i=1}^{n} | p_i - r_i |$$

- Mean Squared Error (*MSE*) places more emphasis on larger deviations

$$MSE = \frac{1}{n}\sum_{i=1}^{n} (p_i - r_i)^2$$

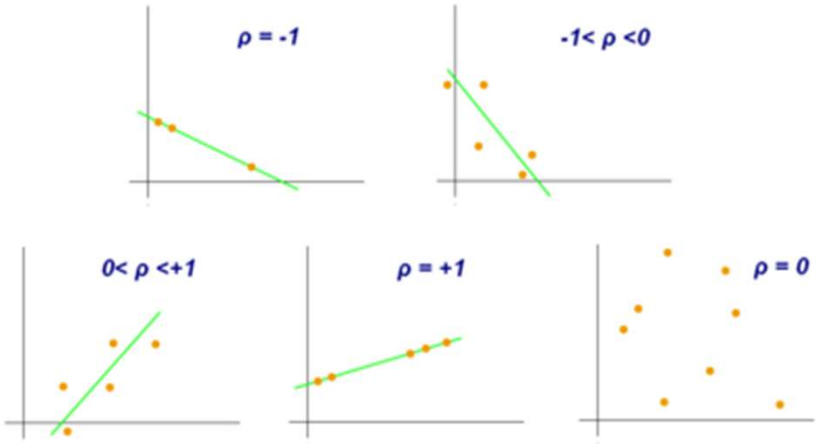
- Root Mean Squared Error (*RMSE*) has similar scale as *MAE* and places more emphasis on larger deviations

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (p_i - r_i)^2}$$

# Metrics for Regression Model Evaluation

– Pearson's Correlation Coefficient (PCC)

- scores well if
  - high actual values get high predictions
  - low actual values get low predictions




$$\mathrm{PCC} = \frac{\sum_{\text{all examples}} (pred - \overline{pred}) \times (act - \overline{act})}{\sqrt{\sum_{\text{all examples}} (pred - \overline{pred})^2} \times \sqrt{\sum_{\text{all examples}} (act - \overline{act})^2}}$$

– R Squared: Coefficient of Determination

- measures the part of the total variation in the dependent variable y that is predictable (explainable) from the explanatory variables X

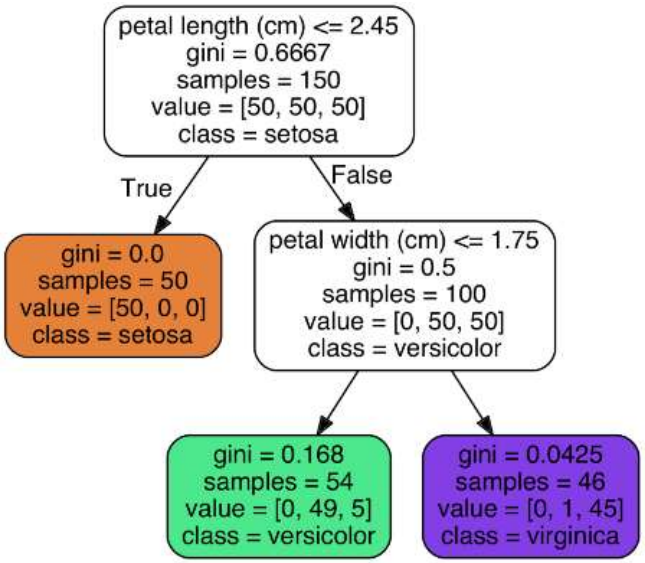
$$R^2 = \frac{\sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

- $R^2 = 1$ : Perfect model as total variation of y can be completely explained from X
- $R^2$ is called ‘squared correlation’ in RapidMiner

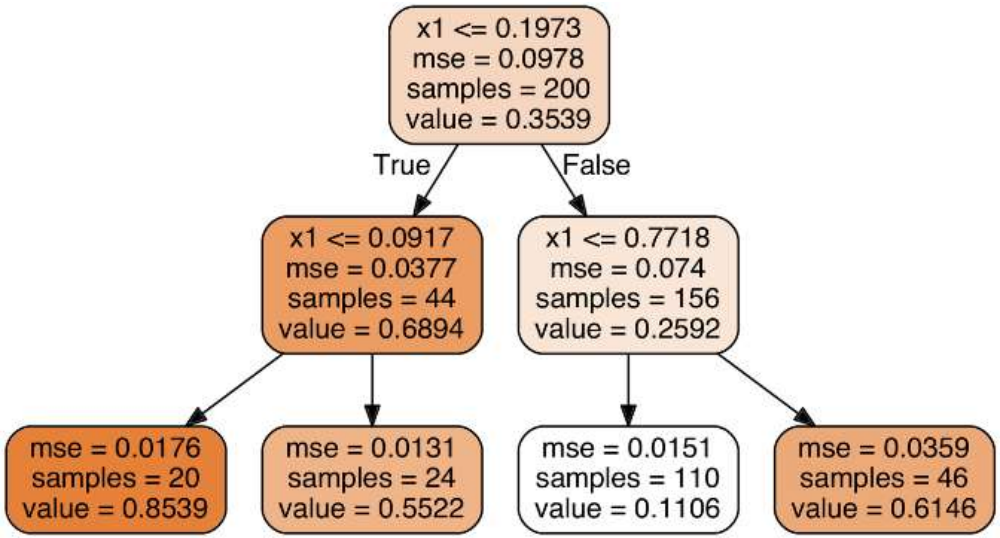
# 4. Regression Trees

- The basic idea of how to learn and apply decision trees can also be used for regression
- Differences:
  1. splits are selected by maximizing the MSE reduction (not GINI or entropy)
  2. prediction is average value of the trainings examples in a specific leaf

## Decision Tree



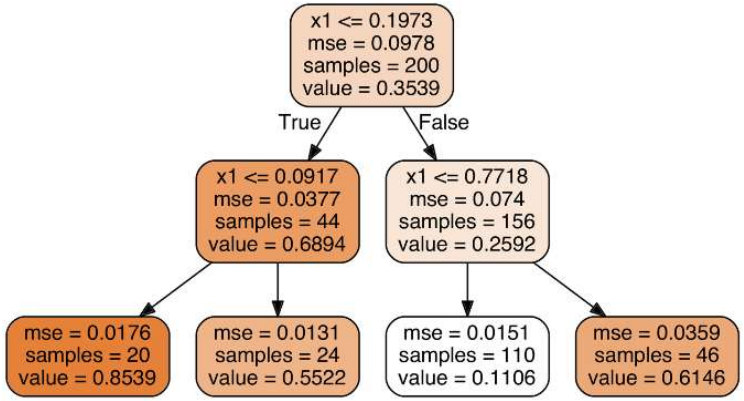

## Regression Tree

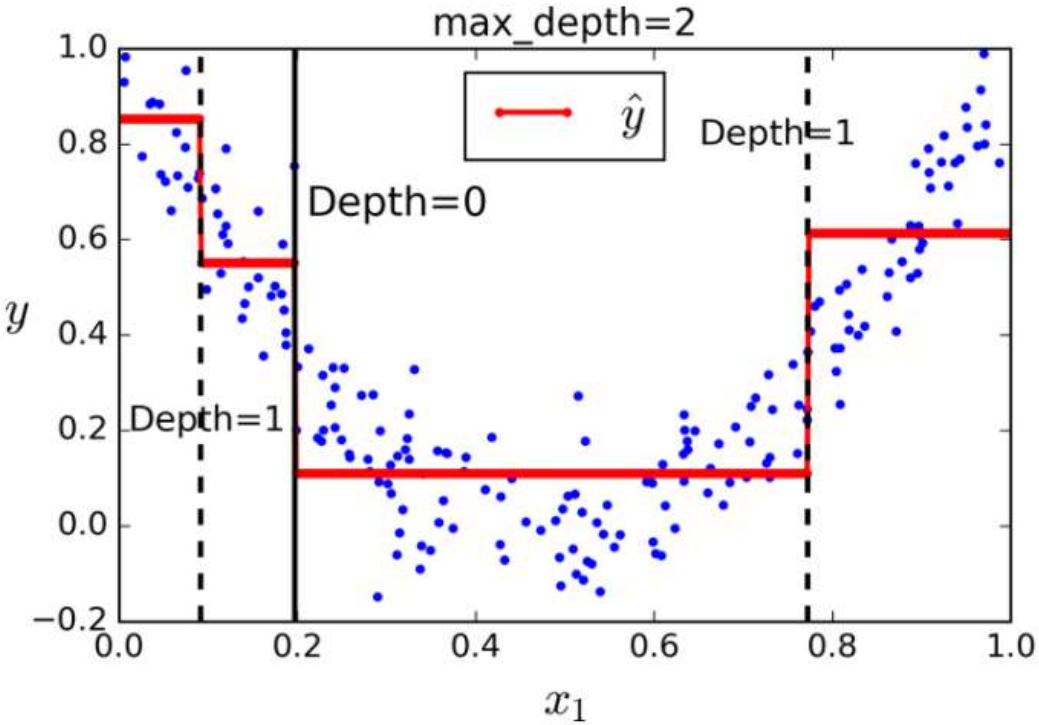


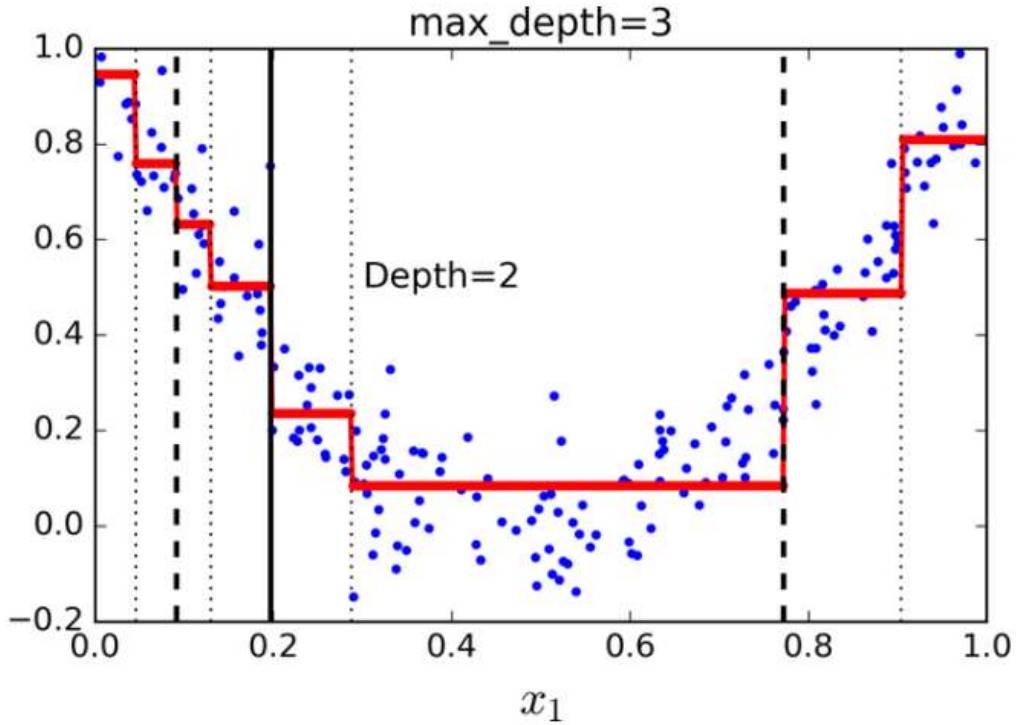
# Regression Trees Fitting the Training Data

Pre-pruning parameters deterime how closely the tree fits the training data

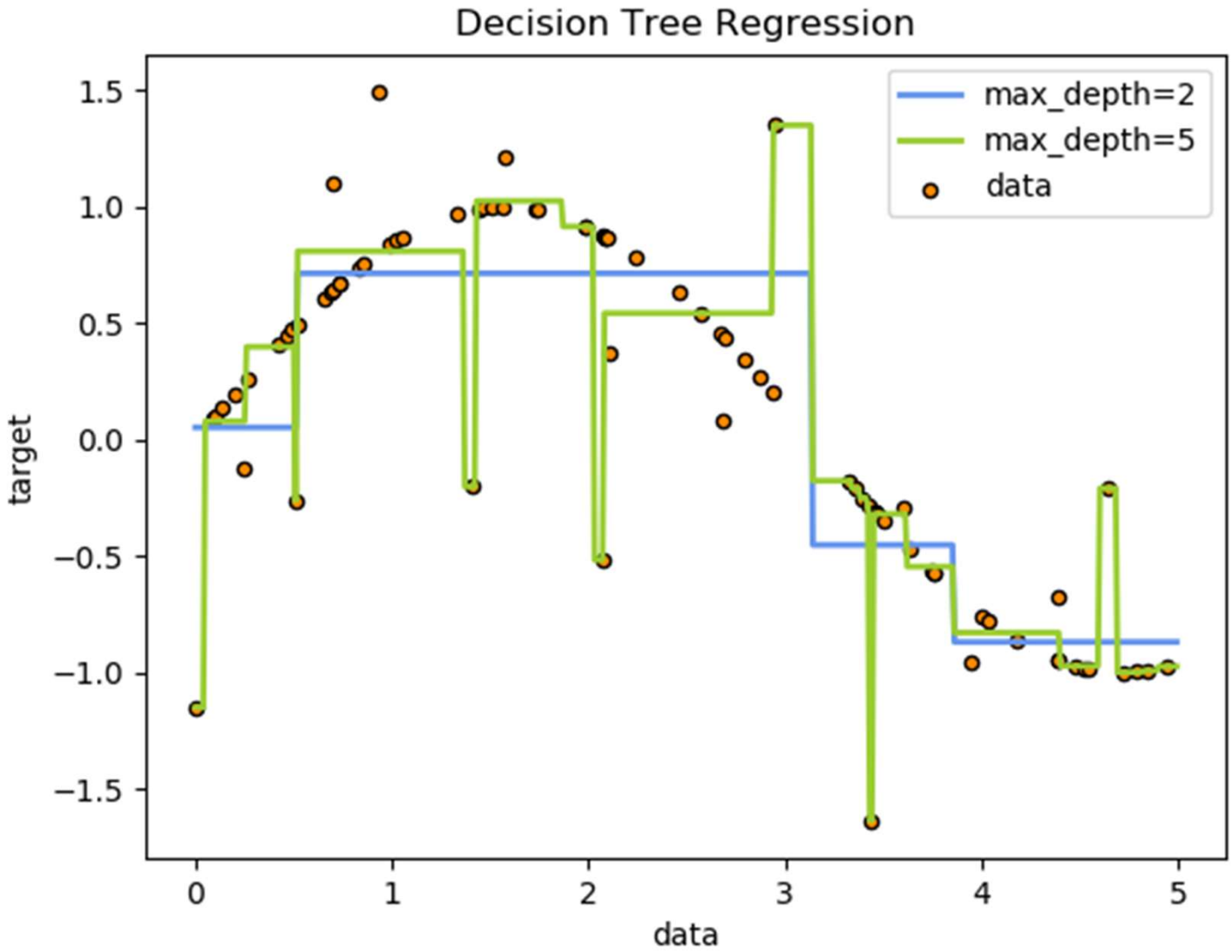
- e.g. max_depth parameter



# Overfitted Regression Tree

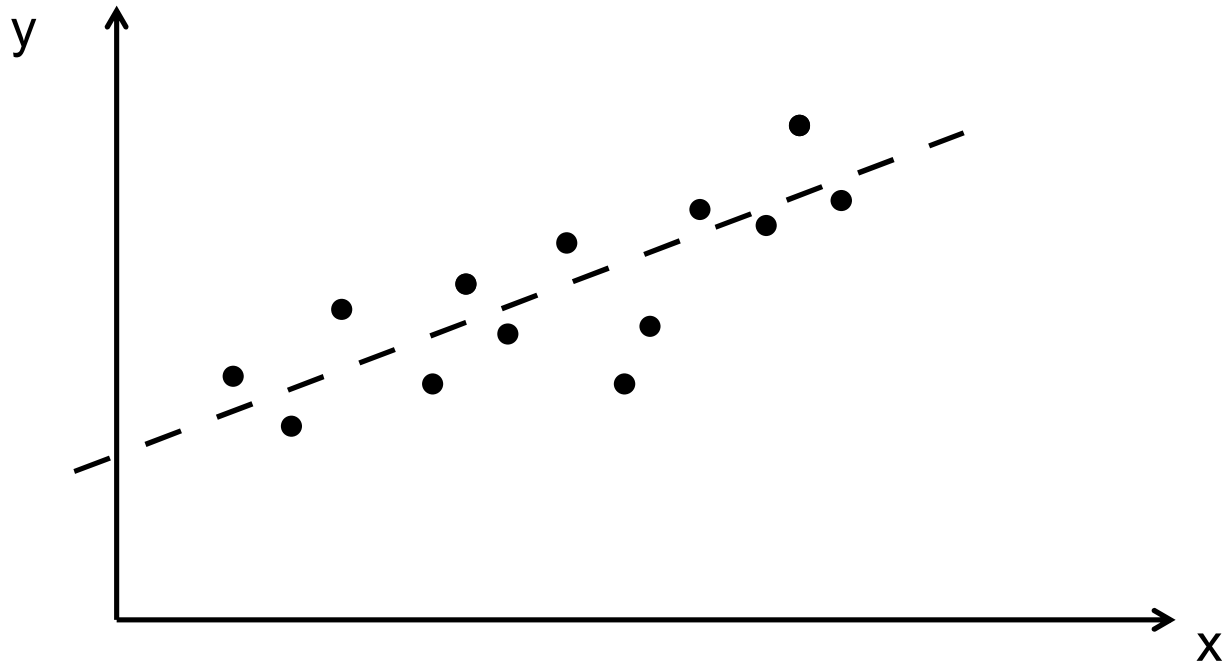



The learning algorithm uses available depth to cover strongest outliers

# 5. Linear Regression

Assumption of Linear Regression: The target variable y is (approximately) linearly dependent on explanatory variables X

- for visualization: we use one variable x (simple linear regression)
- in reality: vector $X = x_1 \ldots x_n$ (multiple linear regression)

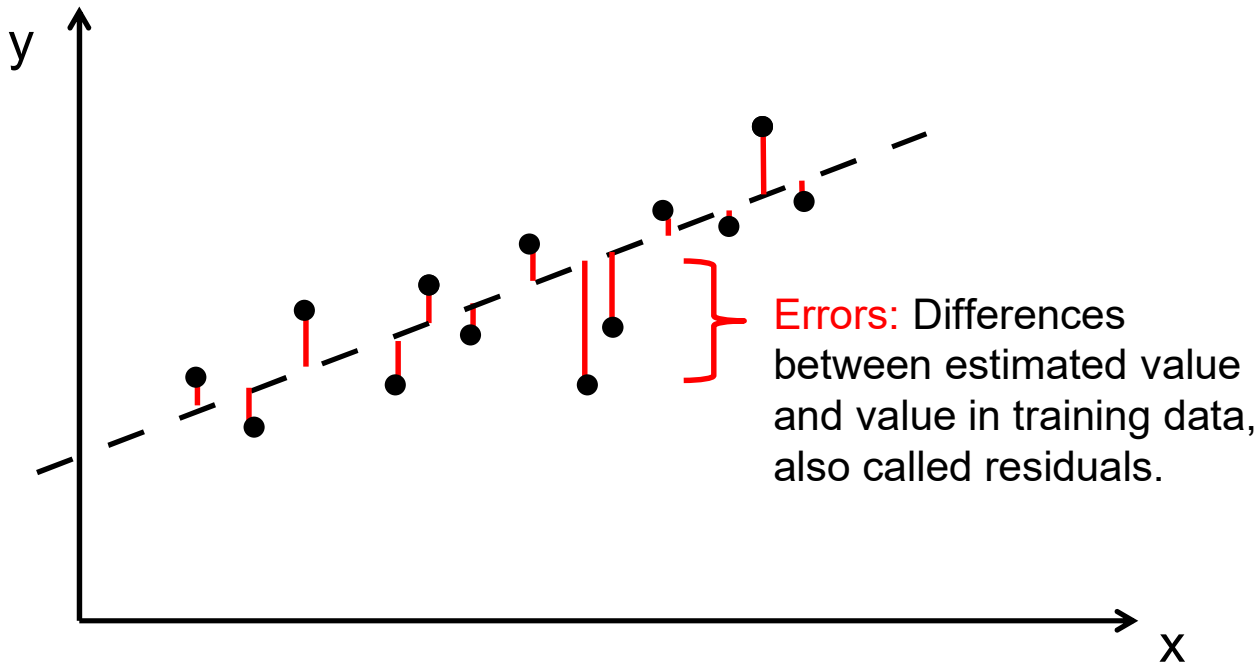


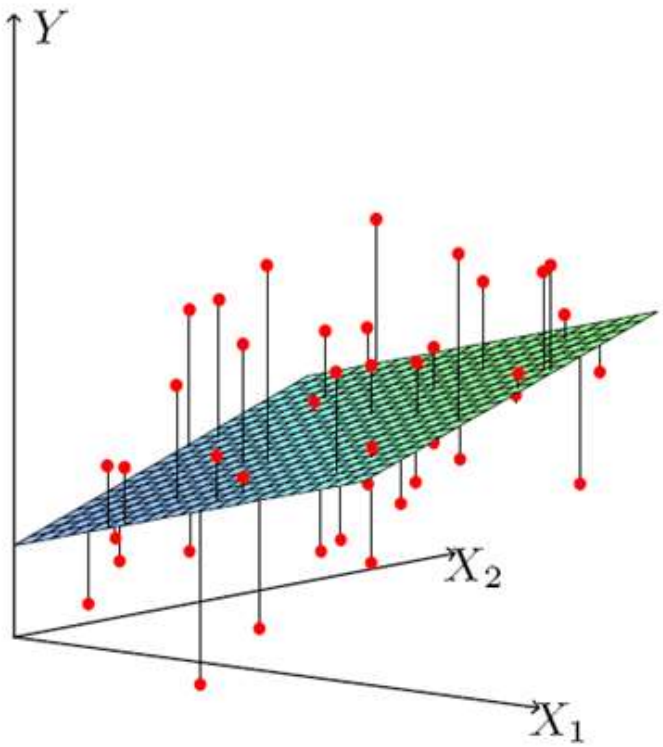

$$\hat{y}(X) = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

# Fitting a Regression Function

Least-Squares Approach: Find the weight vector W = ($w_0$, $w_1$, .., $w_n$) that minimizes the sum of squared error (SSE) for all training examples

$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}(X_i, W))^2$$



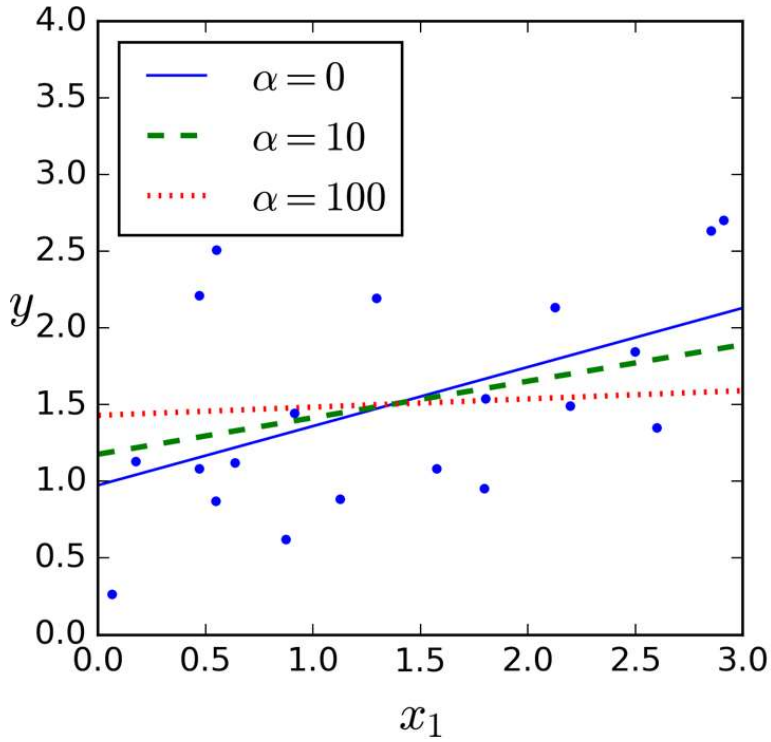



# Ridge Regularization

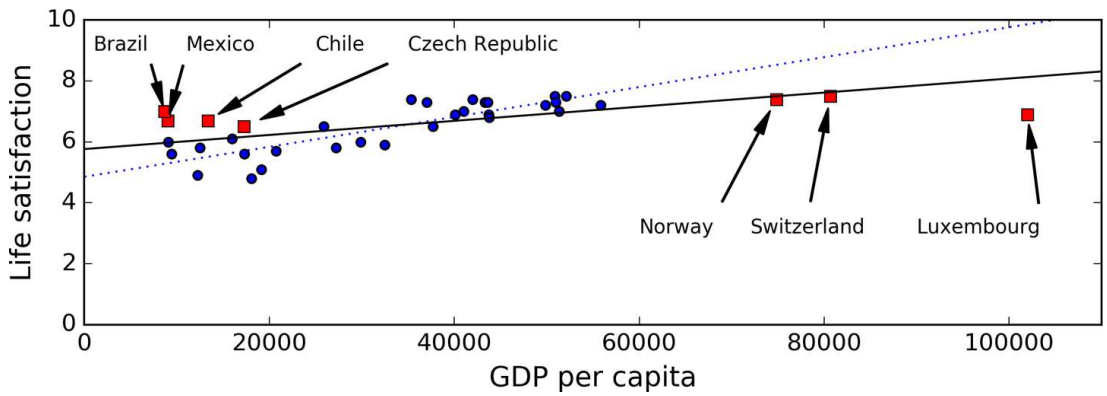
- Variation of least squares approach which tries to avoid overfitting by keeping the weights W small
- Ridge regression cost function to minimize

$$C(W) = MSE(W) + \alpha \sum_{i=1}^{n} w_i^2$$

  - α = 0 : Normal least squares regression
  - α = 100 : Strongly regularized flat curve



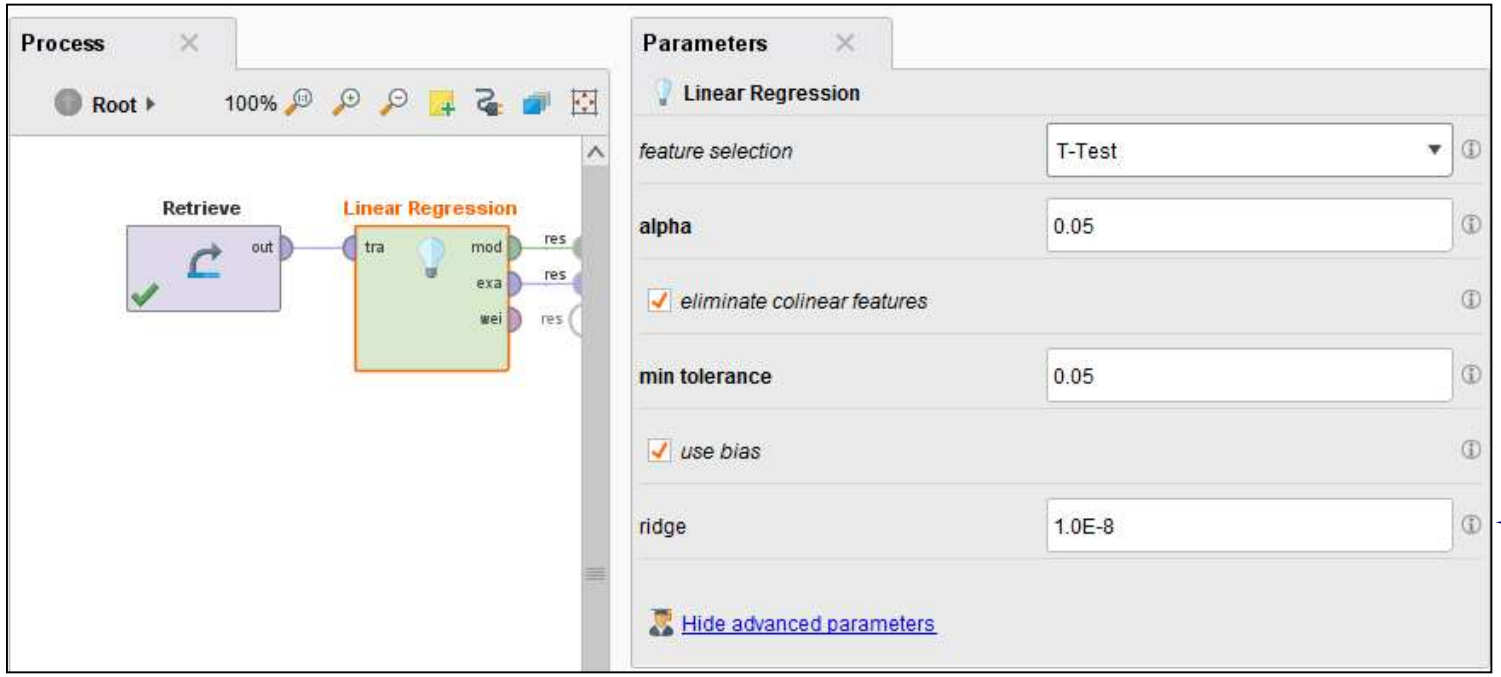

- Example of overfitting due to biased training data



# Feature Selection

- Question: Do all explanatory variables X help to explain y or is only a subset of the variables useful?
- Problem 1: Highly correlated variables (e.g. height in cm and inch)
  - weights are meaningless and one variable should be removed for the better interpretability of the weights
- Problem 2: Insignificant variables (e.g. the weather for stock prices)
  - uncorrelated variables get w=0 or relatively small weights assigned
  - Question for variables having small weights: Is the variable still useful or did it get the weight by chance due to biased training data?
  - Answer: Statistical test with null-hypothesis "w=0 as variable is insignificant"
    - **t-stat:** number of standard deviations that w is away from 0
      - high t-stat ➔ Variable is significant as it is unlikely that weight is assigned by chance
    - **p-value:** Probability of wrongly rejecting the hull-hypothesis
      - p-value close to zero ➔ variable is significant
  - See: James, Witten, et al.: An Introduction to Statistical Learning. Chapter 3.1.2

# Linear Regression in RapidMiner




Choose feature selection method

Correlated features

Ridge (>0) or least squares (0) regression

| Attribute | Coefficient | Std. Error | Std. Coeff... | Tolerance | t-Stat | p-Value | Code |
|---|---|---|---|---|---|---|---|
| a1 | 85.440 | 5.692 | 0.463 | 1.000 | 15.009 | 0 | **** |
| a2 | 127.001 | 5.701 | 0.689 | 0.998 | 22.276 | 0 | **** |
| a3 | 68.118 | 5.730 | 0.369 | 1.000 | 11.888 | 0 | **** |
| a4 | 10.302 | 5.728 | 0.056 | 1.000 | 1.799 | 0.074 | * |
| a5 | -12.452 | 5.723 | -0.068 | 0.999 | -2.176 | 0.031 | ** |
| (Intercept) | 180.418 | 5.671 | ? | ? | 31.814 | 0 | **** |

Feature quality code

p-Value

# Linear Regression in Python

- Two different classes for linear and ridge regression
- Feature selection implemented as separate preprocessing step

Python

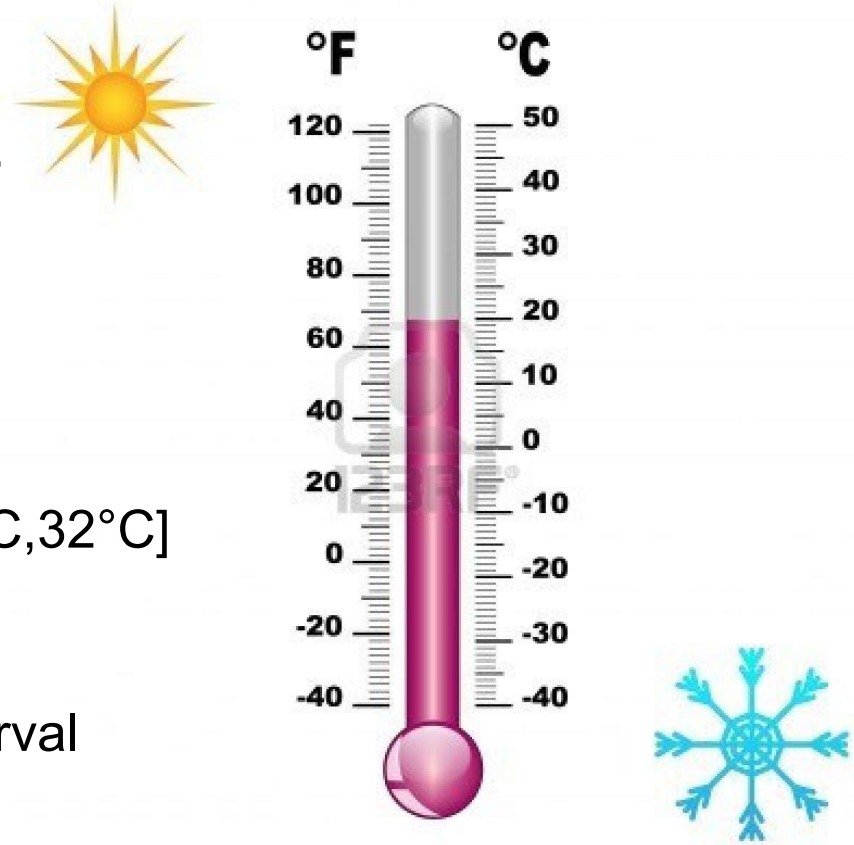
```python
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge

from sklearn.feature_selection import SelectFwe
from sklearn.feature_selection import f_regression

selected_features = SelectFwe(f_regression, alpha=0.05).fit_transform(X, y)
estimator = LinearRegression()
estimator.fit(selected_features, y)
```

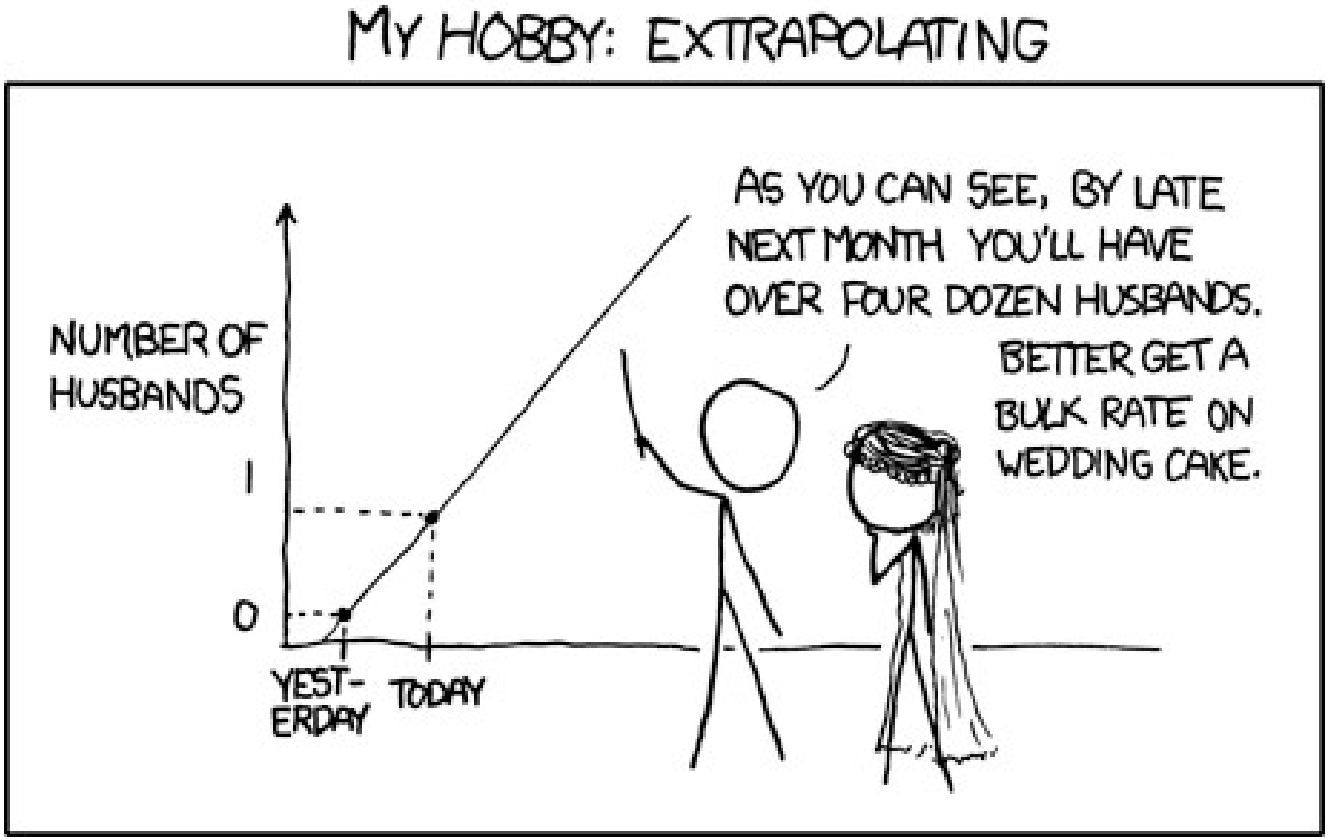
# Interpolation vs. Extrapolation

- Training data:
  - weather observations for current day
  - e.g., temperature, wind speed, humidity, …
  - target: temperature on the next day
  - training values between -15°C and 32°C
- Interpolating regression
  - only predicts values from the interval [-15°C,32°C]
- Extrapolating regression
  - may also predict values *outside* of this interval



# Interpolation vs. Extrapolation

– Interpolating regression is regarded as “safe”
  - i.e., only reasonable/realistic values are predicted




http://xkcd.com/605/

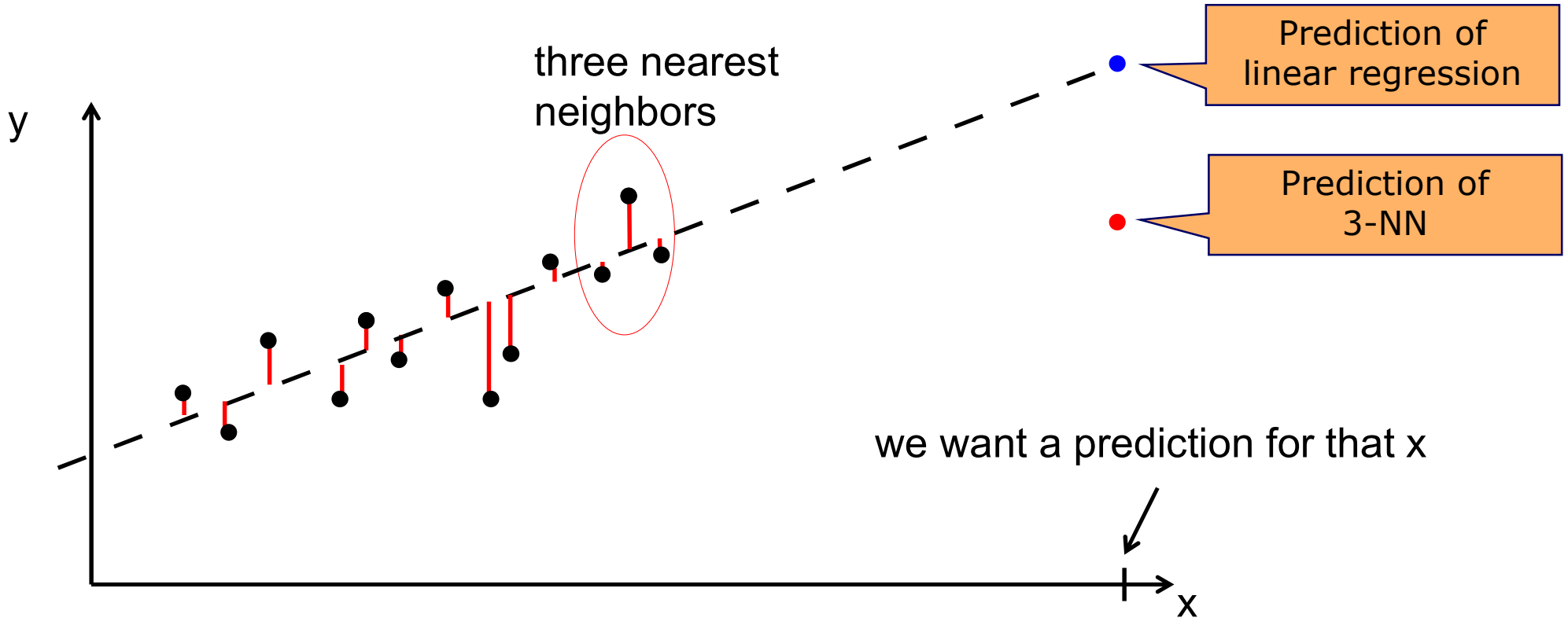
# Interpolation vs. Extrapolation

- Sometimes, however, only extrapolation is interesting
  - how far will the sea level have risen by 2050?
  - how much will the temperature rise in my nuclear power plant?

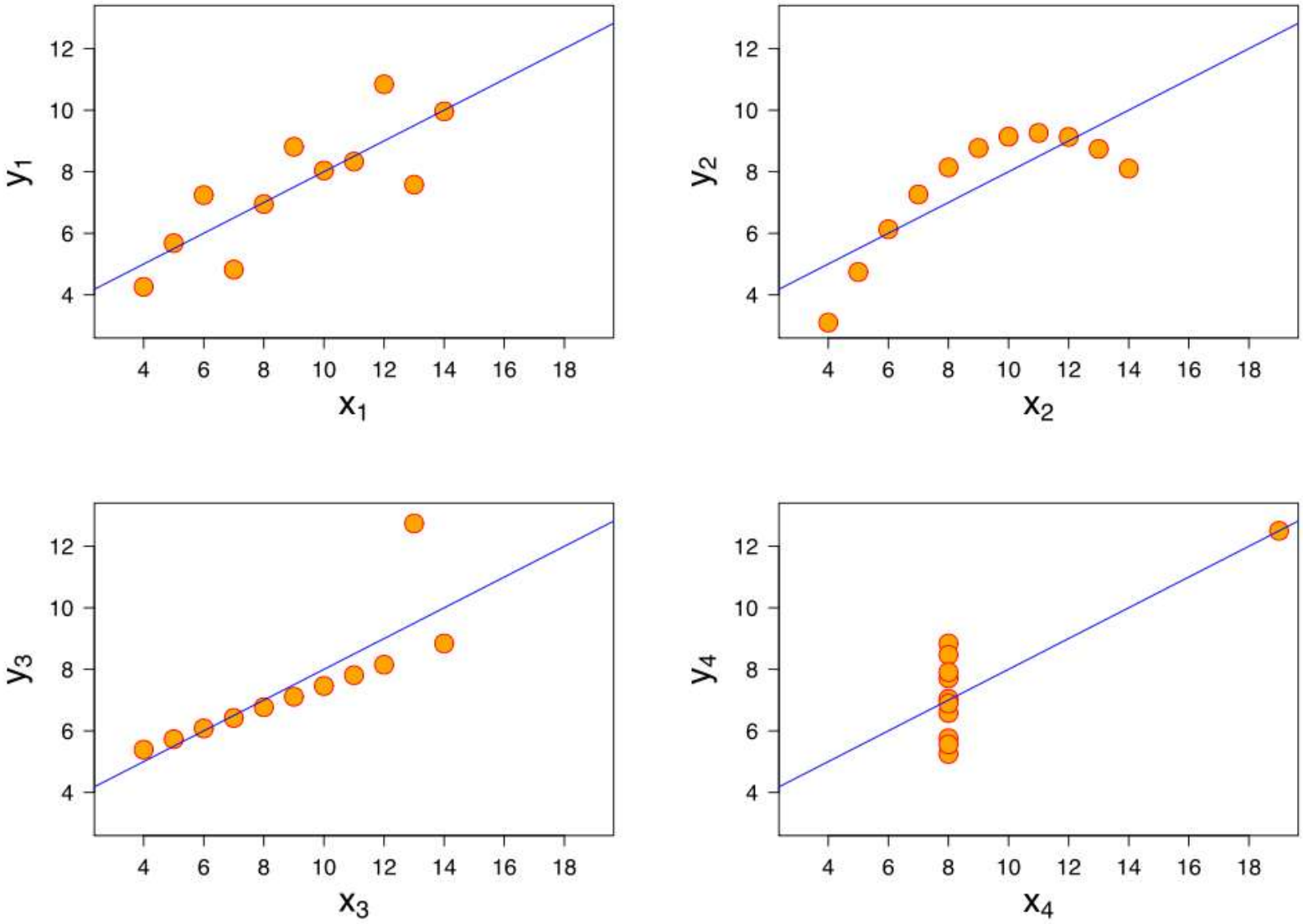
http://i1.ytimg.com/vi/FVfiujbGLfM/hqdefault.jpg

# Linear Regression vs. K-NN Regression

- Linear regression extrapolates
- K-NN and regression trees interpolate

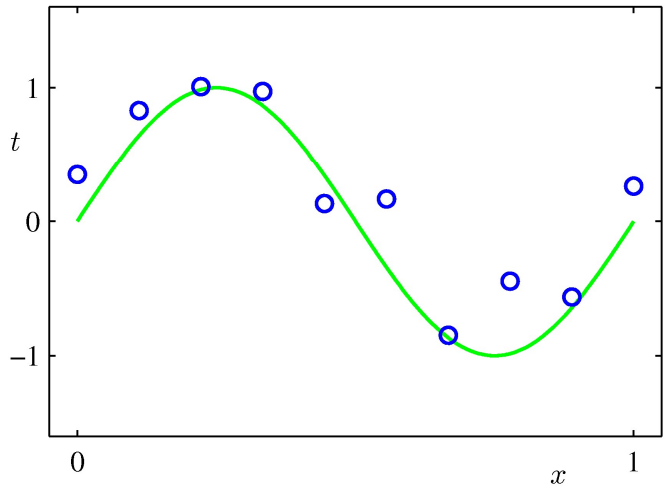


# Linear Regression Examples



# … But What About Non-linear Problems?

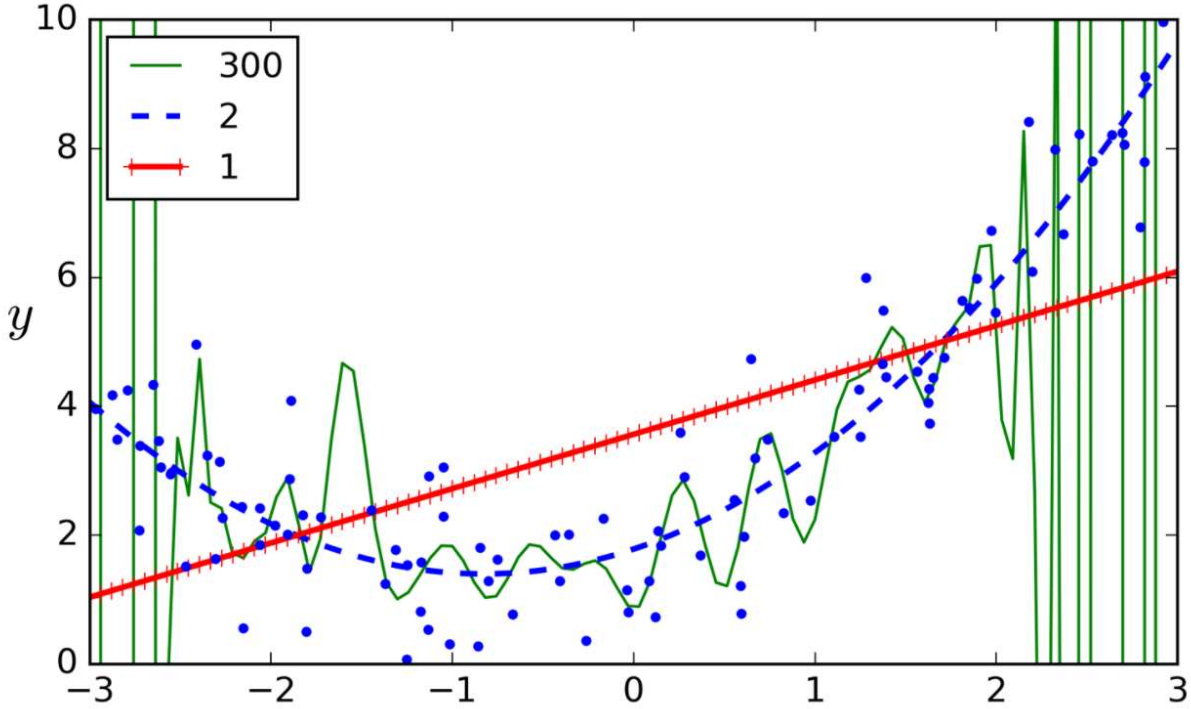
- One possibility is to apply transformations to the explanatory variables X within the regression function
  - e.g. log, exp, square root, square, etc.
    - example: : $\hat{y} = \omega_0 + \omega_1 \cdot x_1^2 + \omega_2 \cdot x_2^2$
  - polynomial transformation
    - example: $\hat{y} = \omega_0 + \omega_1 \cdot x + \omega_2 \cdot x^2 + \omega_3 \cdot x^3$
- This allows use of linear regression techniques to fit much more complicated non-linear datasets



# 6. Polynomial Regression

$$\hat{y}(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

- widely used extension of linear regression
- can also be fitted using the least squares method
- has tendency to over-fit training data for large degrees M
- Workarounds:
  1. decrease M
  2. increase amount of training data



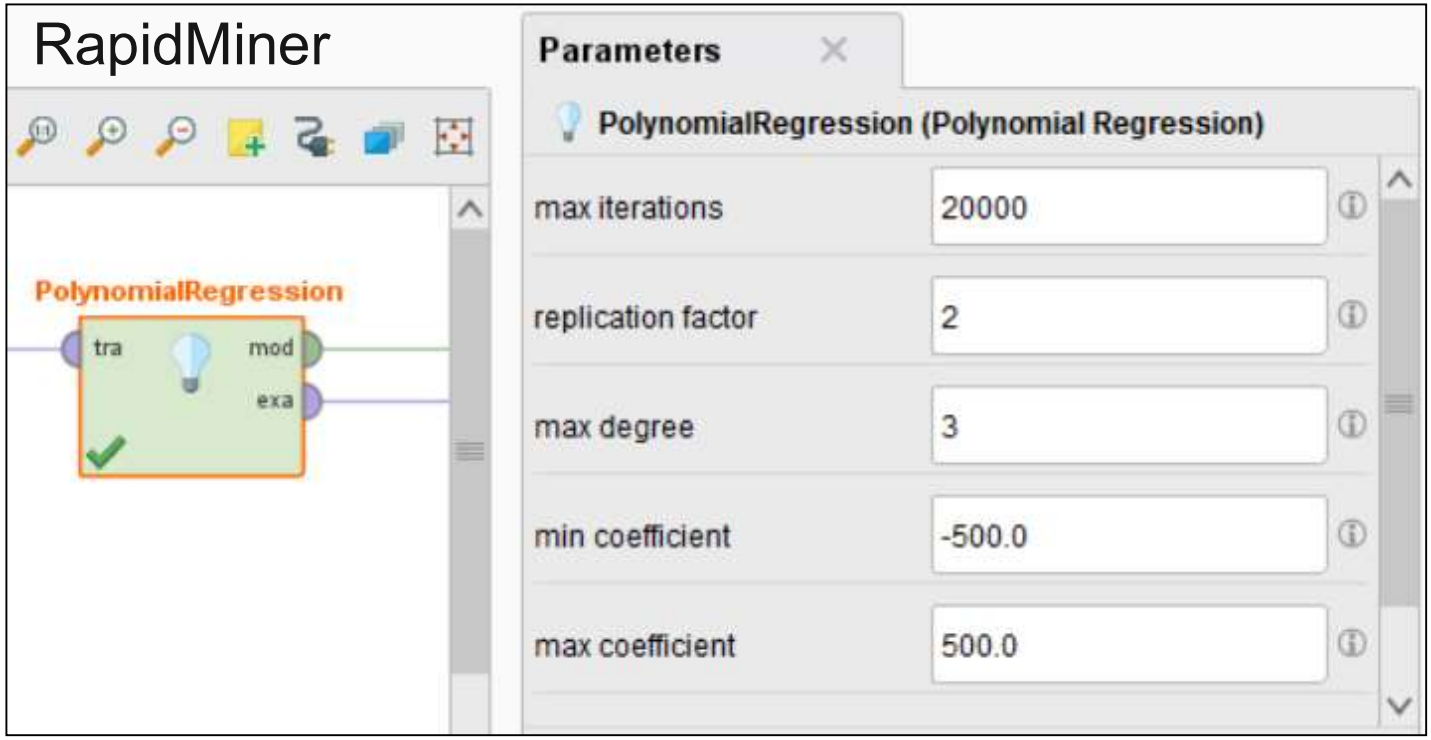
# Polynomial Regression in Python and RapidMiner

## Python

```python
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

poly_features = PolynomialFeatures(degree=2, include_bias=False).fit_transform(X, y)
estimator = LinearRegression()
estimator.fit(poly_features, y)
```

## RapidMiner

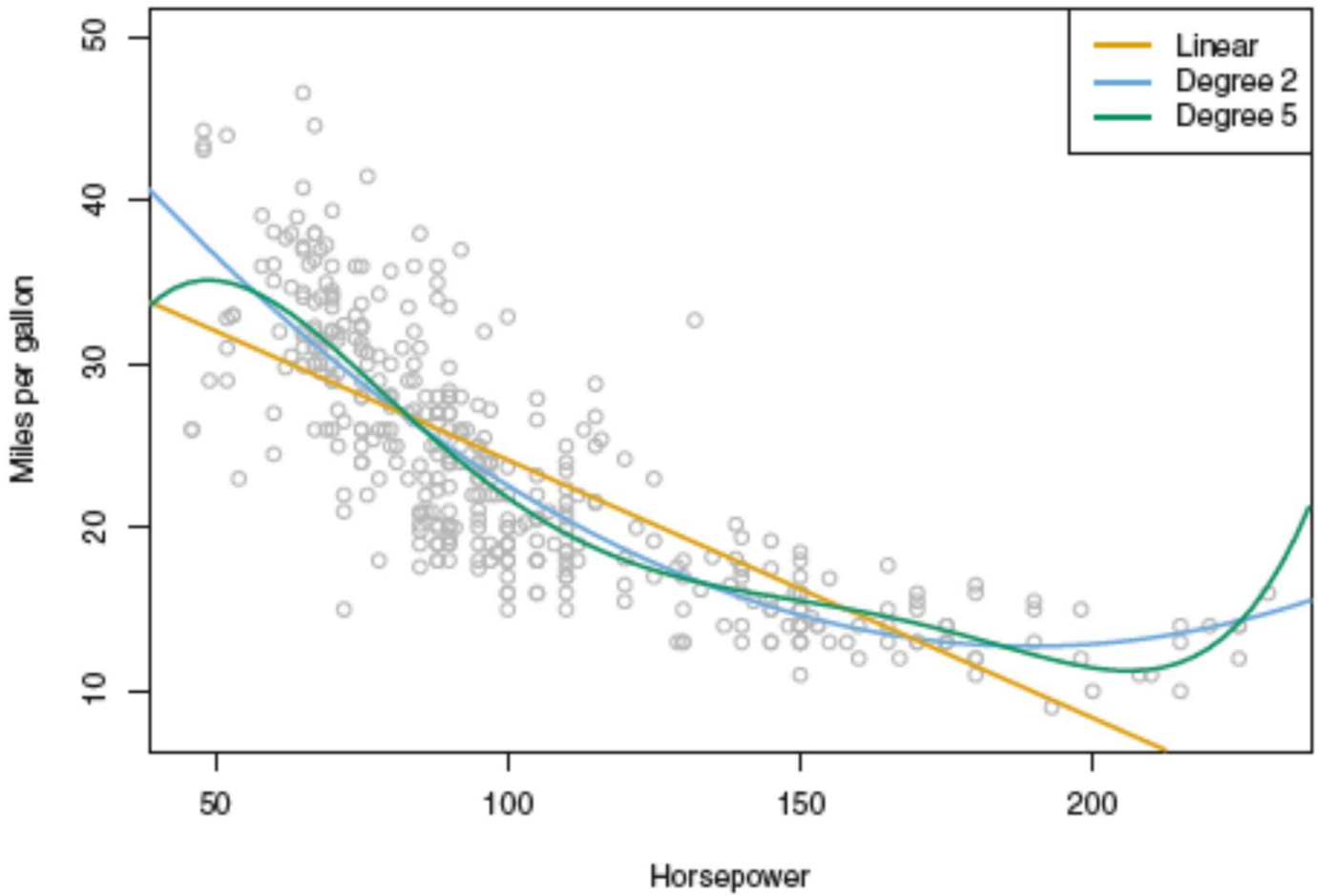



How often may each explanatory variable appear in the function.

Maximal degree to be used in the function

Boundaries for weights W

# Polynomial Regression Overfitting Training Data

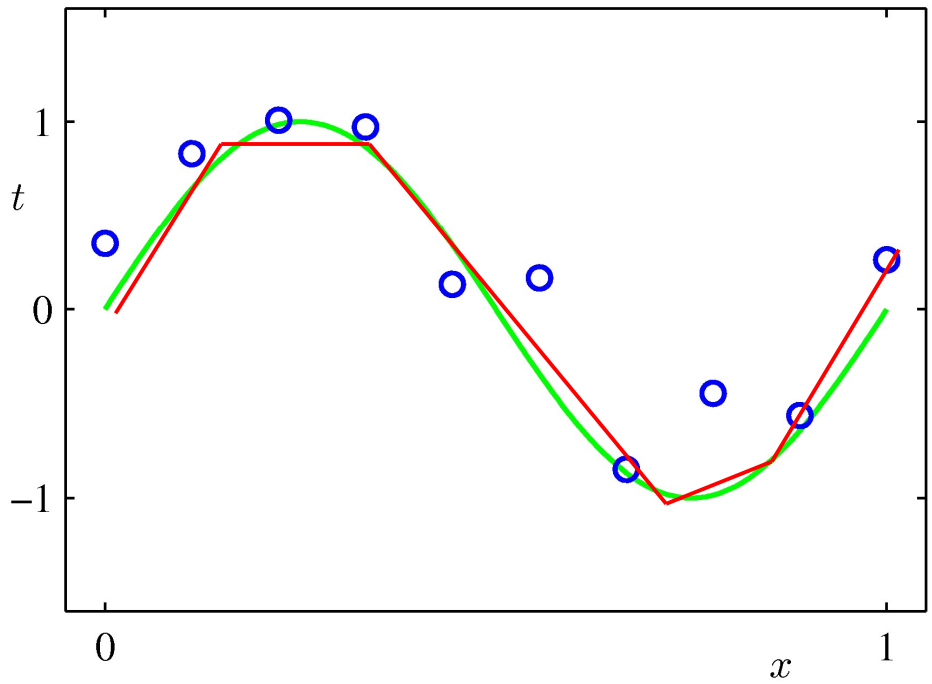



Overfitting often happens in sparse regions

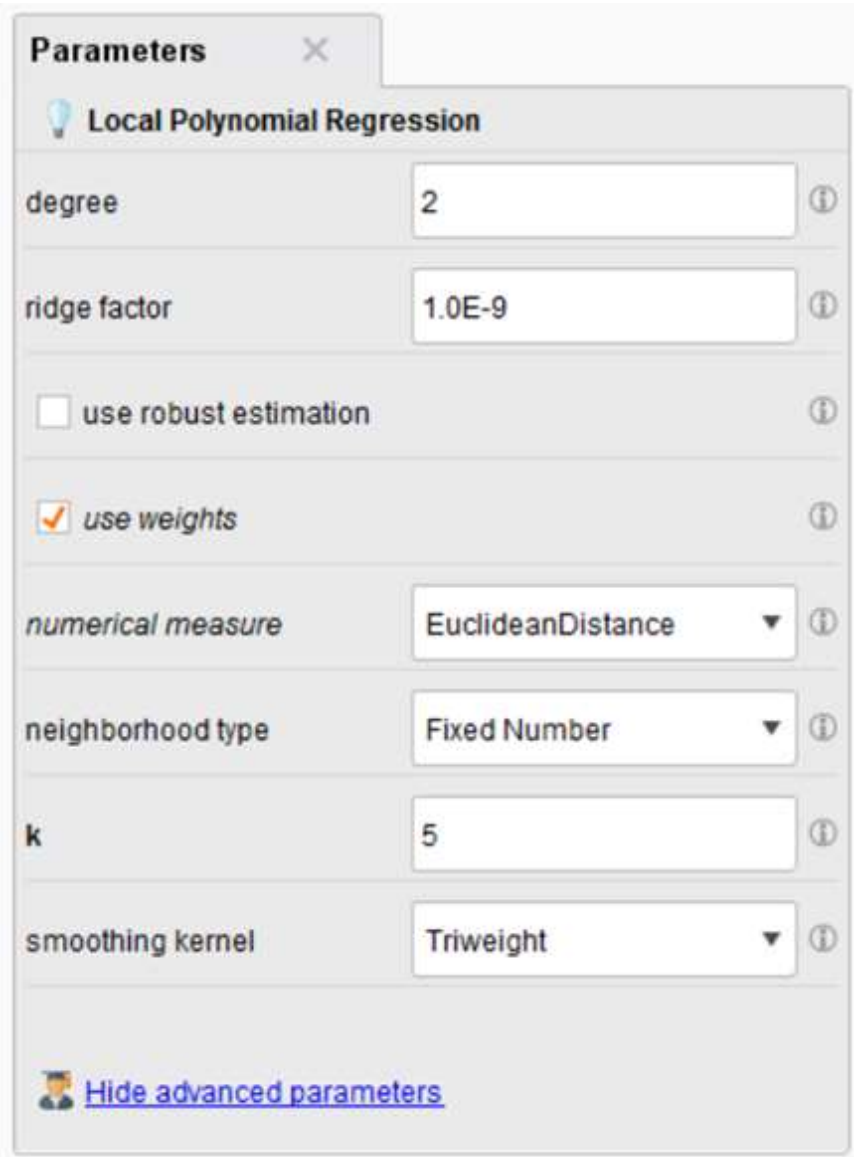
- left and right side of green line
- workaround: Local Regression

# 7. Local Regression

- Assumption: non-linear problems are approximately linear in local areas
- Idea: use linear regression locally for the data point at hand (lazy learning)
- A combination of
  - k nearest neighbors
  - linear regression
- Given a data point
  1. retrieve the k nearest neighbors
  2. learn a regression model using those neighbors
  3. use the learned model to predict y value



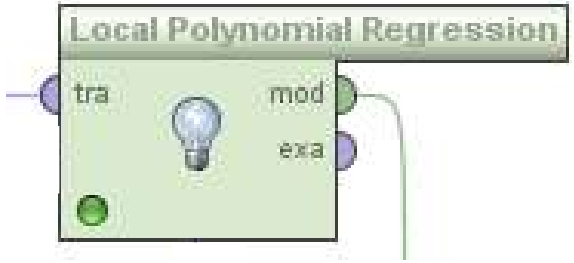
# Local Regression in Rapidminer




Maximal degree to be used. Set to 1 for local linear regression

Distance function for determining neighborhood

Neighborhood size



# Discussion of Local Regression

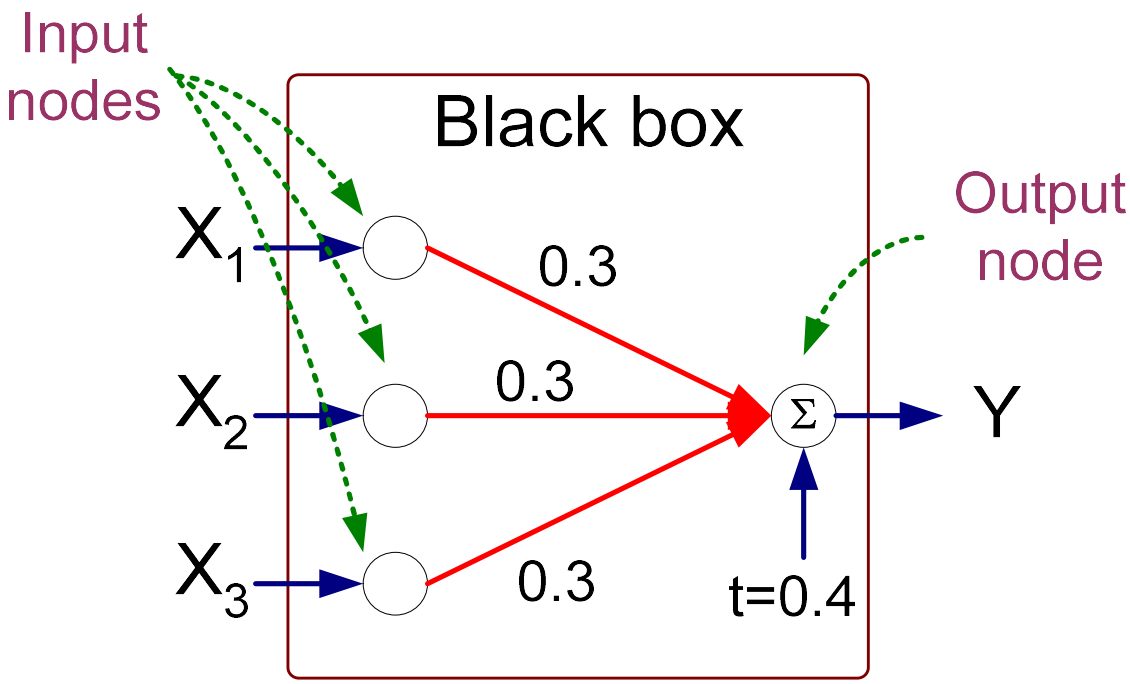
- Advantage: fits non-linear models well
  - good local approximation
  - often better than pure k-NN

- Disadvantage
  - slow at runtime
  - for each test example:
    - find k nearest neighbors
    - compute a local model

# 8. Artificial Neural Networks (ANNs) for Regression

**Recap:** How did we use ANNs for classification?

| $X_1$ | $X_2$ | $X_3$ | Y |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 |




$$Y = I(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 > 0)$$

$$\text{where} \quad I(z) = \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

# Artificial Neural Networks for Regression

- The function *I(z)* was used to separate the two classes:

$$Y = I(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 > 0)$$

$$\text{where} \quad I(z) = \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

- However, we may simply use the inner formula to predict a numerical value (between 0 and 1):

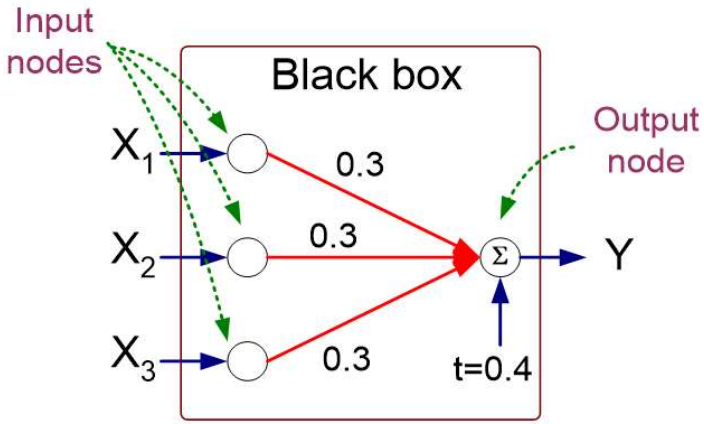
$$\hat{Y} = 0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4$$

- What has changed:
  - we do not use a cutoff for 0/1 predictions, but leave the numbers as they are

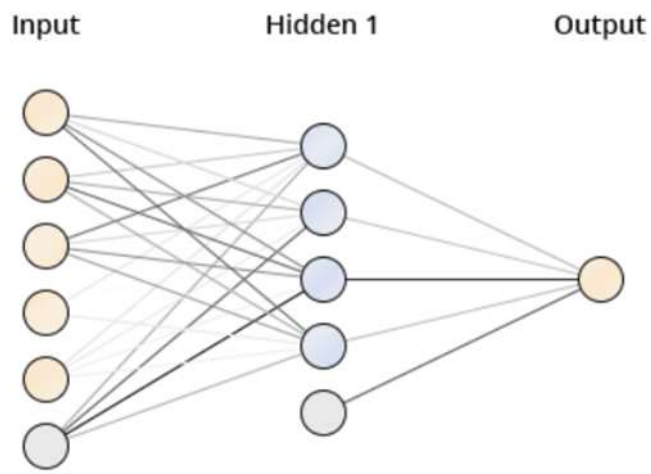
# Artificial Neural Networks for Regression

- Given that our formula is of the form

$$\hat{Y} = 0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4$$

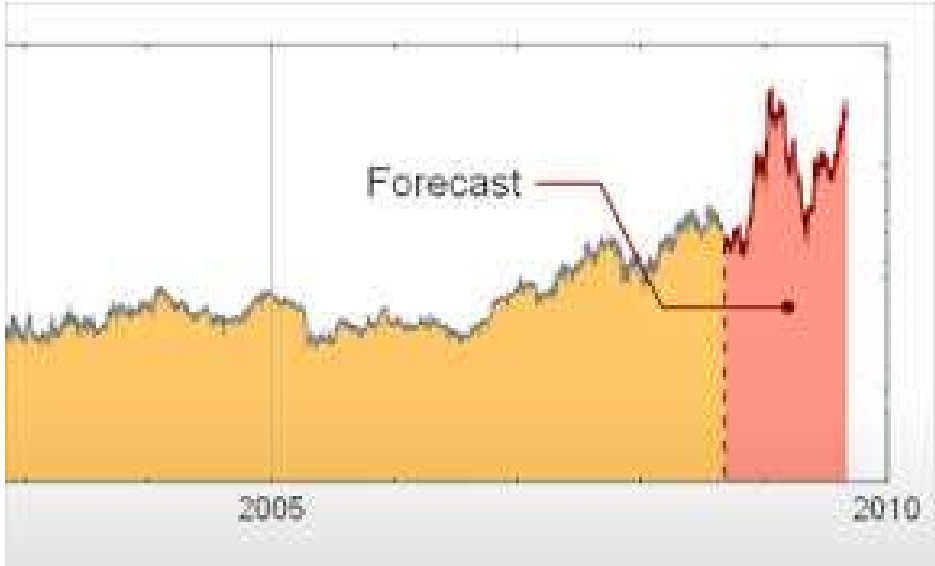



- we can learn only linear models
  - i.e., the target variable is a linear combination the input variables
- More complex regression problems can be approximated
  - by using multiple hidden layers
  - this allows for approximating arbitrary functions
- Deep ANNs take this idea further by
  - employing millions of neurons
  - arranging them into specific network topologies
- If you use ANNs be cautions about overfitting!



# 9. Time Series Forecasting

- A **time series** is a series of data points indexed in time order
  - examples: Stock market prices, ocean tides, birth rates, temperature
- **Forecasting**: Given a time series, predict data points that continue the series into the future
  - explicitly deals with time, which is not explicitly considered by other regression techniques
  - aims to predict future values of the same variable
- Approaches:
  1. Data-driven: Smoothing
  2. Model-Driven:
     1. component models of time series
     2. other regression techniques combined with windowing



# Smoothing

## Simple Moving Average (SMA)

– average of the last n values, as more recent values might matter more

$$m_{\mathrm{MA}}^{(n)}(t) = \frac{1}{n}\sum_{i=0}^{n-1} x(t-i)$$

## Exponential Moving Average (EMA)

– exponentially decrease weight of older values

$$m_{\mathrm{EMA}}^{(n)}(t) = \alpha \cdot x(t) + (1-\alpha) \cdot m_{\mathrm{EMA}}^{(n)}(t-1)$$

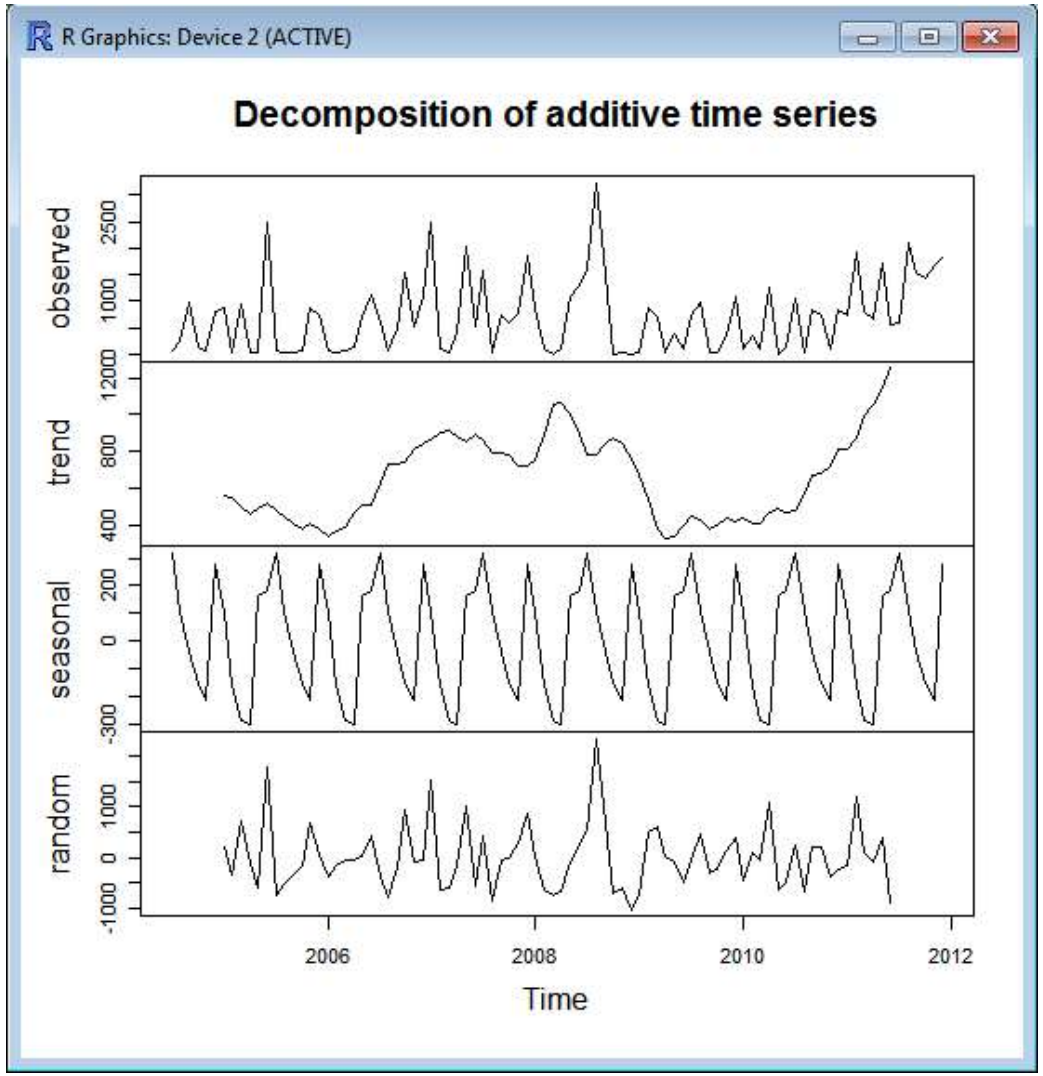

DAX: red = SMA(38 days), yellow = SMA(200 days)

# Component Models of Time Series

Assume **time series** to consist of four components:

1. Long-term trend ($T_t$)
2. Cyclical effect ($C_t$)
3. Seasonal effect ($S_t$)
4. Random variation ($R_t$)

Series = $T_t + C_t + S_t + R_t$

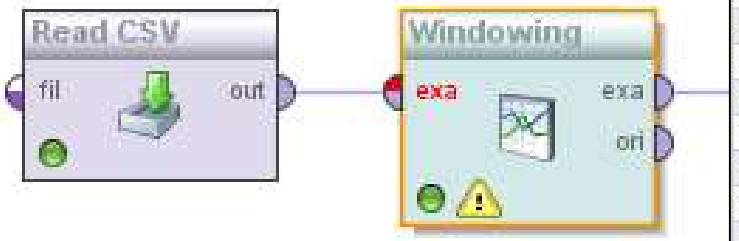


# Windowing

- Idea: Transformation of forecasting problem into “classical” learning problem
  - either regression or classification
  - by only taking the last k time periods into account
- Example: Weather forecasting
  - using the weather from the three previous days
  - Possible model:
    - sunny, sunny, sunny → sunny
    - sunny, cloudy, rainy → rainy
    - sunny, cloudy, cloudy → rainy
- Assumption:
  - only the last k time periods matter for the forecast
  - the older past is irrelevant

# Windowing in RapidMiner and Python

Series Extension




Result Overview | ExampleSet (Windowing)

Data View | Meta Data View | Plot View | Advanced Charts | Annotations

ExampleSet (250 examples, 2 special attributes, 6 regular attributes)

| Row No. | Date | Weather-2 | Weather-1 | Weather-0 | Temperature-2 | Temperature-1 | Temperature-0 | label |
|---|---|---|---|---|---|---|---|---|
| 1 | 04.01.2013 | sunny | cloudy | cloudy | 23 | 24 | 28 | cloudy |
| 2 | 07.01.2013 | cloudy | cloudy | cloudy | 24 | 28 | 32 | rainy |
| 3 | 08.01.2013 | cloudy | cloudy | rainy | 28 | 32 | 19 | sunny |
| 4 | 09.01.2013 | cloudy | rainy | sunny | 32 | 19 | 24 | rainy |
| 5 | 10.01.2013 | rainy | sunny | rainy | 19 | 24 | 25 | cloudy |
| 6 | 11.01.2013 | sunny | rainy | cloudy | 24 | 25 | 17 | sunny |
| 7 | 14.01.2013 | rainy | cloudy | sunny | 25 | 17 | 14 | sunny |
| 8 | 15.01.2013 | cloudy | sunny | sunny | 17 | 14 | 12 | rainy |
| 9 | 16.01.2013 | sunny | sunny | rainy | 14 | 12 | 26 | sunny |
| 10 | 17.01.2013 | sunny | rainy | sunny | 12 | 26 | 23 | cloudy |
| 11 | 18.01.2013 | rainy | sunny | cloudy | 26 | 23 | 24 | cloudy |
| 12 | 21.01.2013 | sunny | cloudy | cloudy | 23 | 24 | 28 | cloudy |
| | | | | | 24 | 28 | 32 | rainy |
| | | | | | 28 | 32 | 19 | sunny |
| | | | | | 32 | 19 | 24 | rainy |
| | | | | | 19 | 24 | 25 | cloudy |
| | | | | | 24 | 25 | 17 | sunny |

Result Overview | RuleModel (Rule Induction)

Text View | Annotations

## RuleModel

```
if Temperature-0 ≤ 21 and Temperature-2 > 20.500 then sunny  (0 / 0 / 62)
if Temperature-1 > 21 and Temperature-1 ≤ 27 then cloudy  (81 / 0 / 0)
if Weather-2 = cloudy then rainy  (0 / 62 / 0)
else sunny  (0 / 0 / 18)

correct: 223 out of 223 training examples.
```

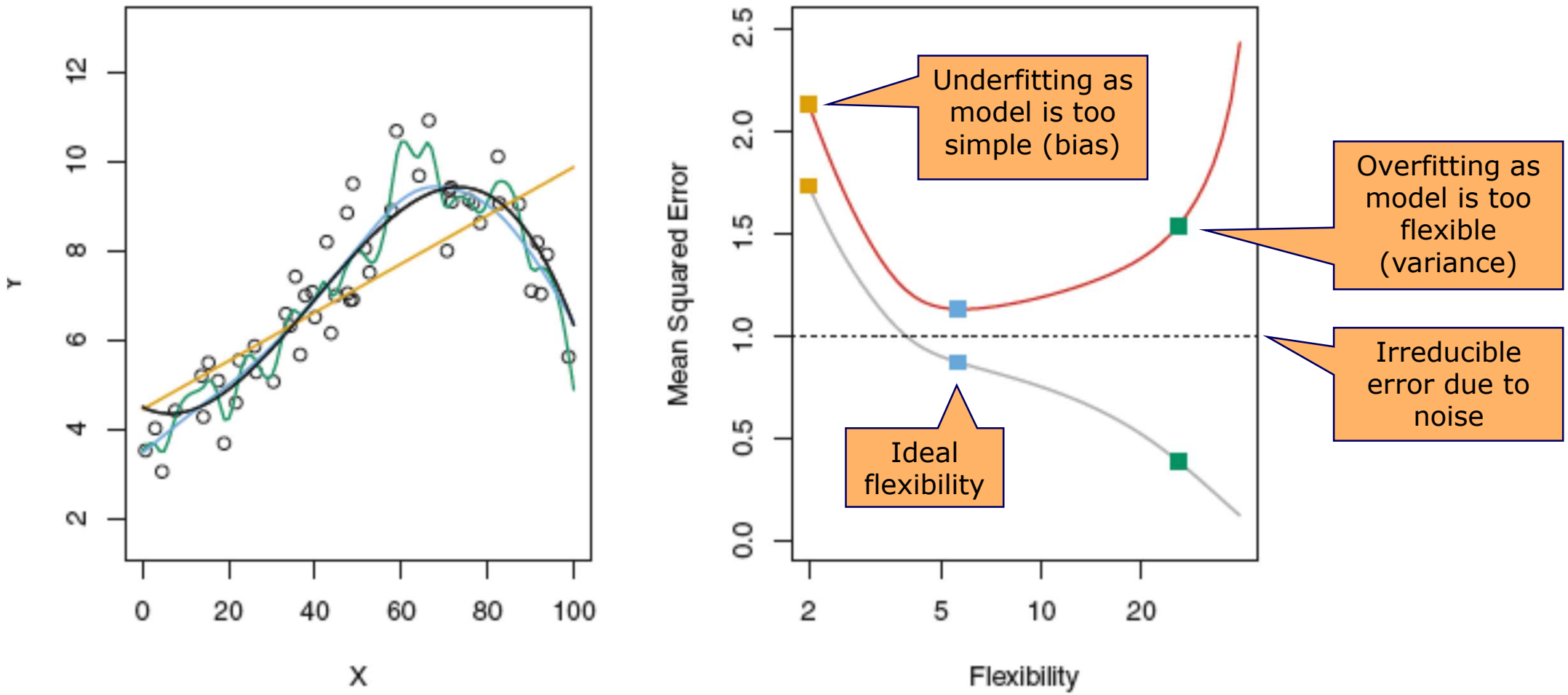
**Python:** Windowing is implemented using
pandas.DataFrame.shift()
pandas.DataFrame.rolling()

# 10. The Bias/Variance-Tradeoff

- We want to learn regression as well as classification models that generalize well to unseen data
- The generalization error of any model can be understood as a sum of three errors:
  1. **Bias:** Part of the generalization error due to wrong model complexity
     - simple model (e.g. linear regression) used for complex real world phenomena
     - model thus underfits the training and test data
  2. **Variance:** Part of the generalization error due to a model's excessive sensitivity to small variations in the training data
     - models with high degree of freedom/flexibility (like polynomial regression models or deep trees) are likely to overfit the training data
  3. **Irreducible Error:** Error due to noisiness of the data itself
     - to reduce this part of the error the training data needs to be cleansed (by removing outliers, fixing broken sensors)

# The Bias/Variance-Tradeoff

- Left: Three models with different flexibility trying to fit a function
  - Orange: Linear regression. Green, blue: Polynomials of different degrees
- Right: Training error (gray) and test error (red) in relation to model flexibility



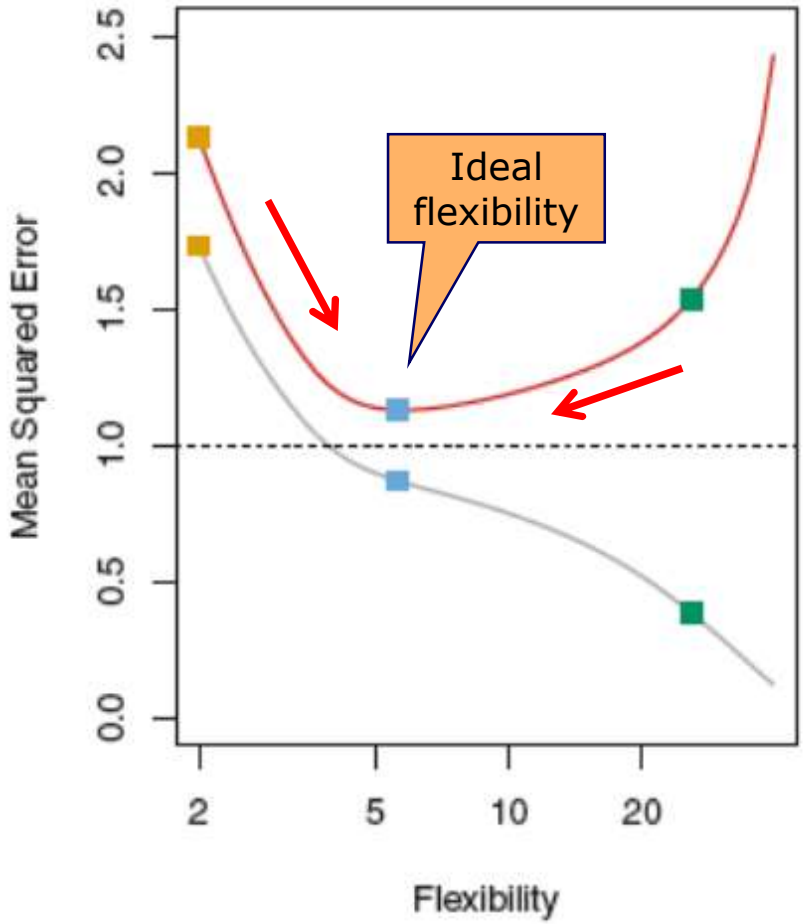
# Learning Method and Hyperparameter Selection

We try to find the ideal flexibility (bias/variance-tradeoff) by

1. Testing different learning methods
   - Linear regression, polynomial regression, …
   - Decision Trees, ANNs, Naïve Bayes, …
2. Testing different hyperparameters
   - degree of polynomial, ridge
   - max depth of tree, min examples branch
   - number of hidden layers of ANN

But we have three more options:

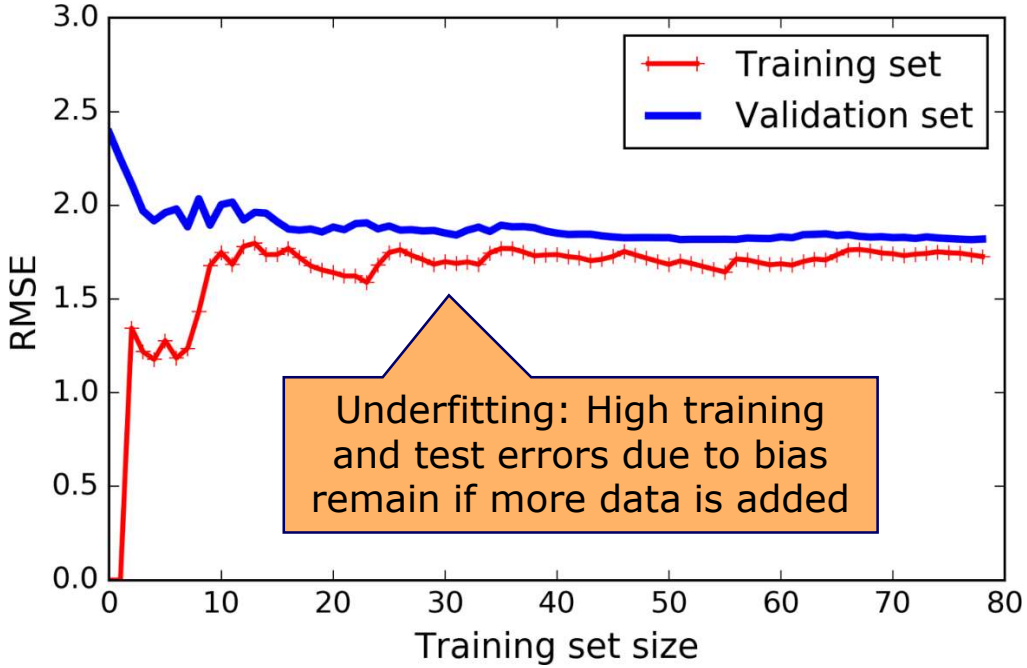
1. increase the amount of training data
2. increase the interestingness of the data by including more corner cases
3. cleanse the training data



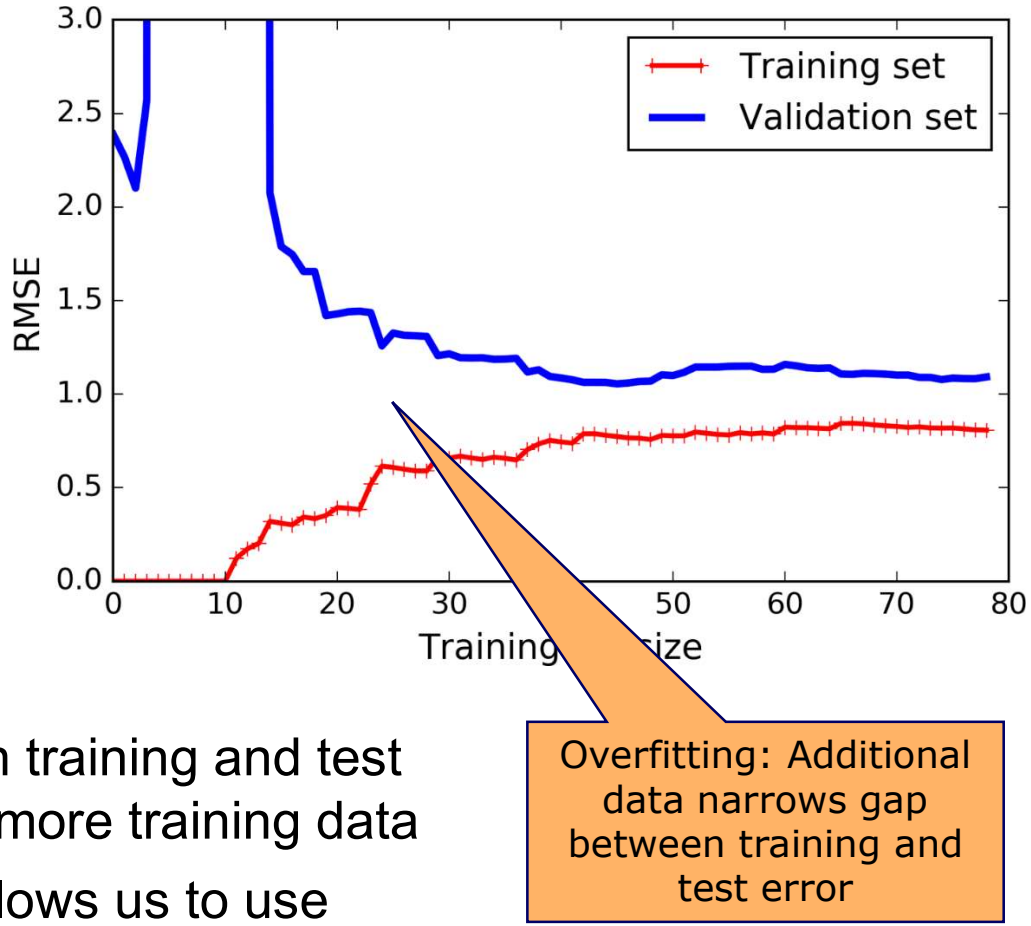
# Learning Curves

– Visualize the training error and test error for different training set sizes

Low Flexibility Model (e.g. Linear regression)




High Flexibility Model (e.g. Polynomial regression)




– For overfitting models, the gap between training and test error can often be narrowed by adding more training data

– Thus, having more training data also allows us to use models having a higher flexibility, e.g. Deep Learning

# Summary

- Regression
  - predict numerical values instead of classes
- Model evaluation
  - metrics: (root) mean squared error, R squared, …
  - methods: (nested) cross-validation
- Methods
  - k nearest neighbors, regression trees, artificial neural networks
  - linear regression, polynomial regression, local regression
  - time series prediction
- For good performance on unseen data
  - choose learning method having the right flexibility (bias/variance-tradeoff)
  - use large quantities of interesting training data

# Literature

- Solving practical regression tasks using:
  - RapidMiner: Kotu: Predictive Analytics - Chapter 5, 10
  - Python: Geron: Hands-on Machine Learning - Chapter 4
- Sophisticated coverage of regression including theoretical background
  - James, Witten, et al.: An Introduction to Statistical Learning Chapters 3, 7, 8