Data Science 2

Vorlesung 2 - Intro

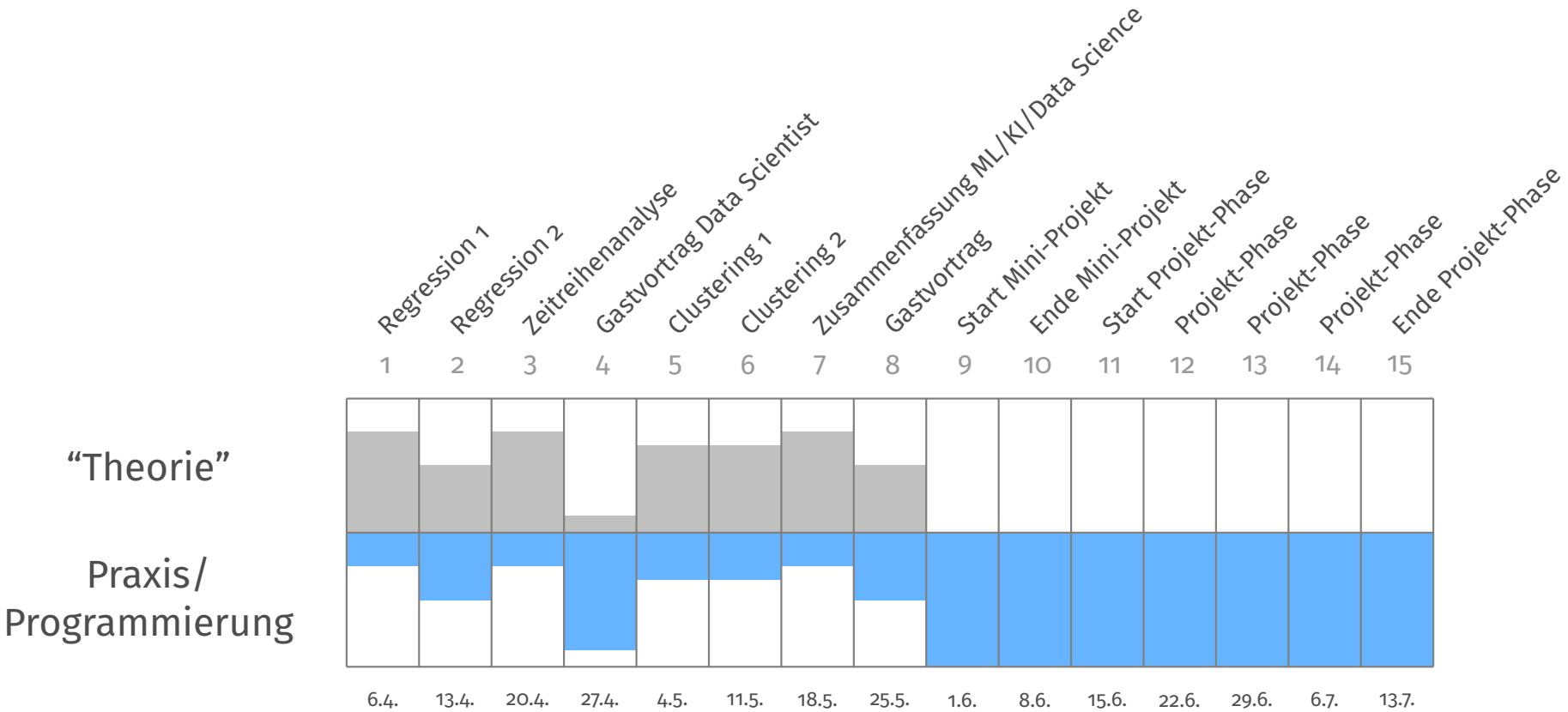
Prof. Dr. Christian Bockermann

Hochschule Bochum

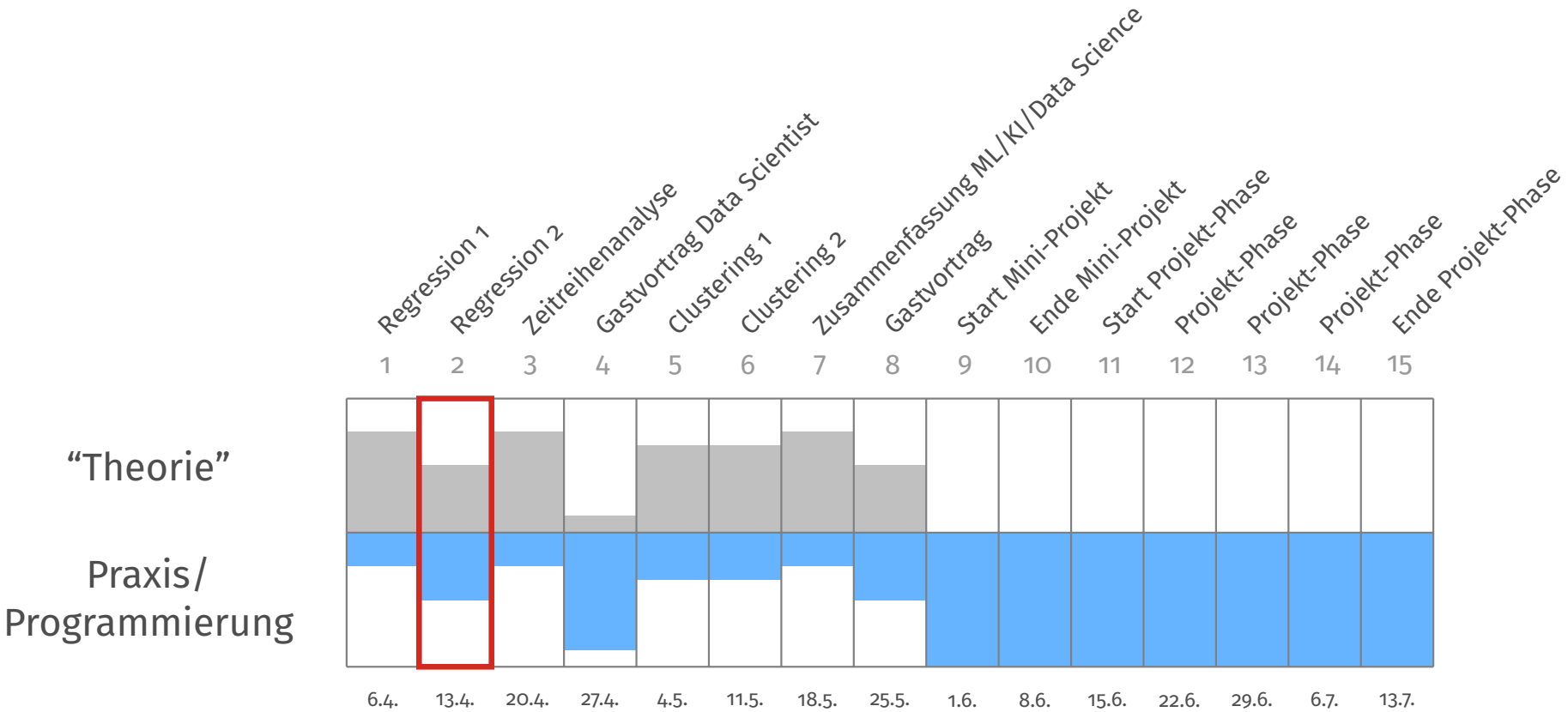
Sommersemester 2021

Inhalt

Themen der Vorlesung



Themen der Vorlesung



Wiederholung - lineare Regression

Lineare Regression

Einfache Modell-Klasse linearer Funktionen

$$\begin{aligned}
\hat{f}(\mathbf{x}) &= b + \sum_{i=1}^{p} w_i x_i \\
&= \sum_{i=0}^{p} w_i x_i \quad \text{mit } w_0 = b, x_0 = 1 \\
&= \mathbf{w}^T \mathbf{x} = \hat{f}_{\mathbf{w}}(\mathbf{x})
\end{aligned}$$

Lineare Regression

Einfache Modell-Klasse linearer Funktionen

$$\begin{aligned}\hat{f}(\mathbf{x}) &= b + \sum_{i=1}^{p} w_i x_i \\ &= \sum_{i=0}^{p} w_i x_i \quad \text{mit } w_0 = b, x_0 = 1 \\ &= \mathbf{w}^T \mathbf{x} = \hat{f}_{\mathbf{w}}(\mathbf{x})\end{aligned}$$

Parameter $\mathbf{w} = (w_0, w_1, \ldots, w_p)$ bestimmen das Modell

Lineare Regression

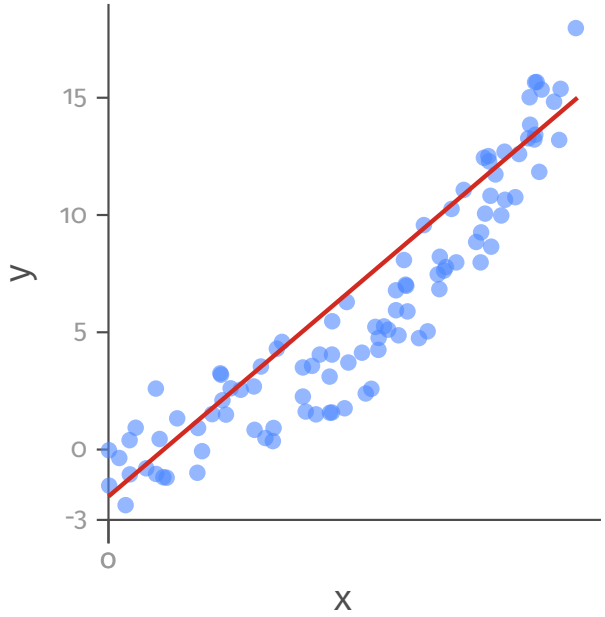
Einfache Modell-Klasse linearer Funktionen

$$\begin{aligned}\hat{f}(\mathbf{x}) &= b + \sum_{i=1}^{p} w_i x_i \\ &= \sum_{i=0}^{p} w_i x_i \quad \text{mit } w_0 = b, x_0 = 1 \\ &= \mathbf{w}^T \mathbf{x} = \hat{f}_{\mathbf{w}}(\mathbf{x})\end{aligned}$$

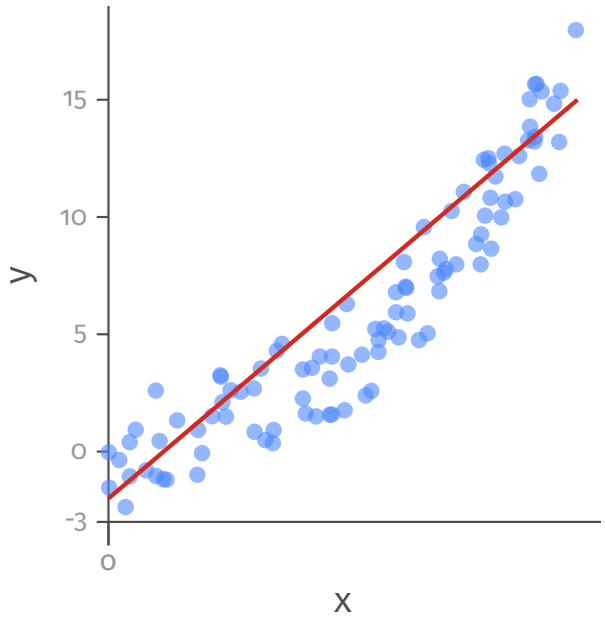
Parameter $\mathbf{w} = (w_0, w_1, \ldots, w_p)$ bestimmen das Modell

Idee: Bestimme w so, dass der Trainingsfehler minimal wird!

Was ist, wenn die Daten nicht linear erklärbar sind?



Was ist, wenn die Daten nicht linear erklärbar sind?



Funktionsklasse linearer Funktionen
reicht nicht immer aus!

Polynome für Multiple Regression

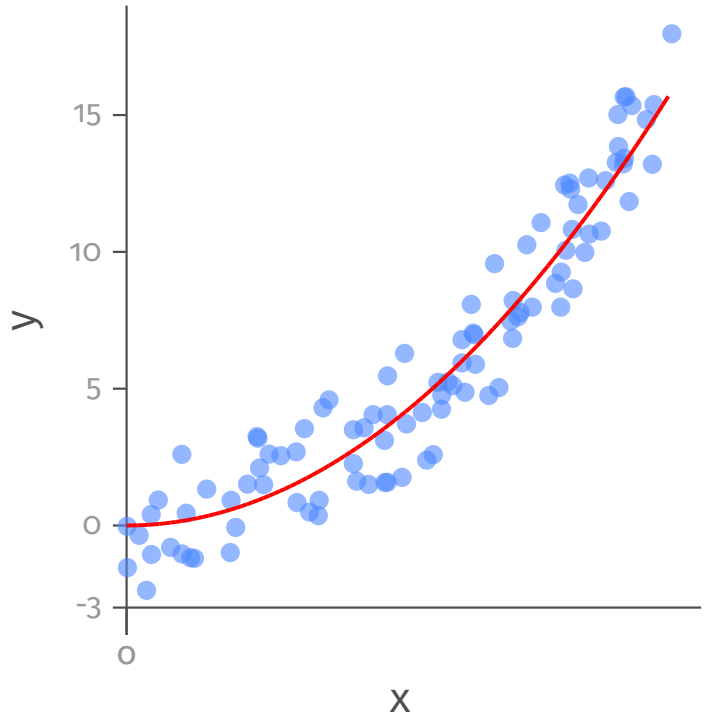
Erweiterung auf mehrere Merkmale:

$$x_1, x_2 \Rightarrow x_1, x_2, x_1x_2, x_1^2, x_2^2$$

x_1	x_2
0.445	0.259
0.158	0.528
0.487	0.561
0.755	0.884
0.495	0.312

x_1	x_2	x_1x_2	x_1^2	x_2^2
0.445	0.259	0.115	0.198	0.067
0.158	0.528	0.083	0.025	0.278
0.487	0.561	0.274	0.237	0.315
0.755	0.884	0.668	0.571	0.781
0.495	0.312	0.154	0.245	0.097

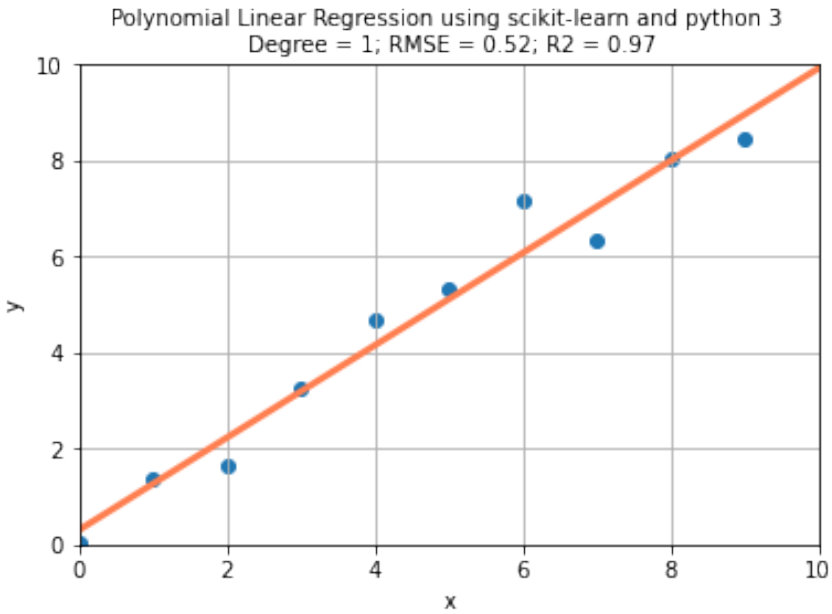
Polynom 2-ten Grades



Generierte Daten und das Polynom $q(x) = 4x^2$

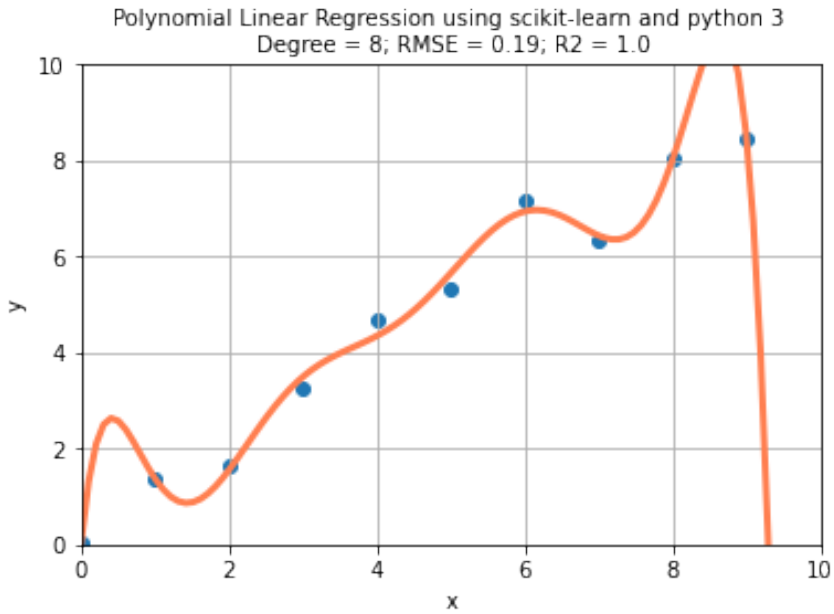
Overfitting, Modell-Komplexität

Optimierung auf geringen Trainingsfehler



$Err_{rmse} = 0.52$

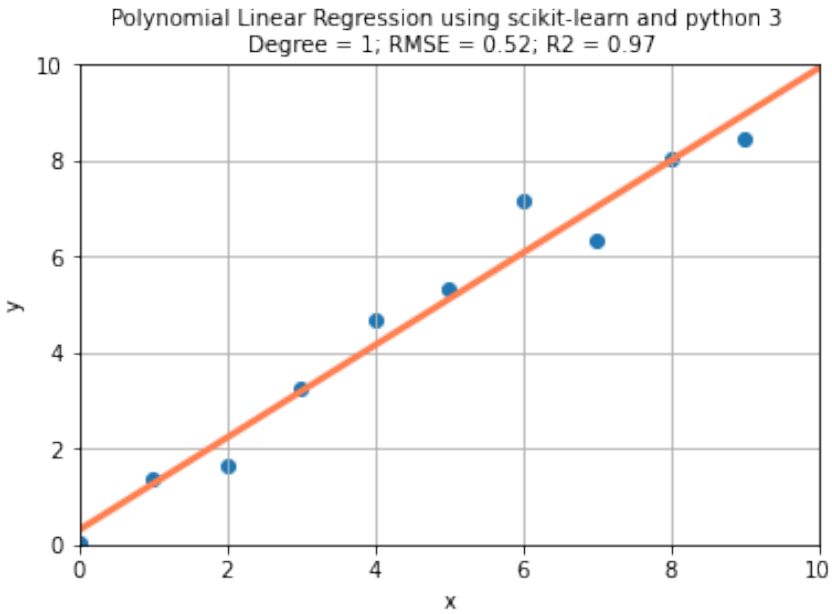
Niedrige Komplexität



$Err_{rmse} = 0.19$

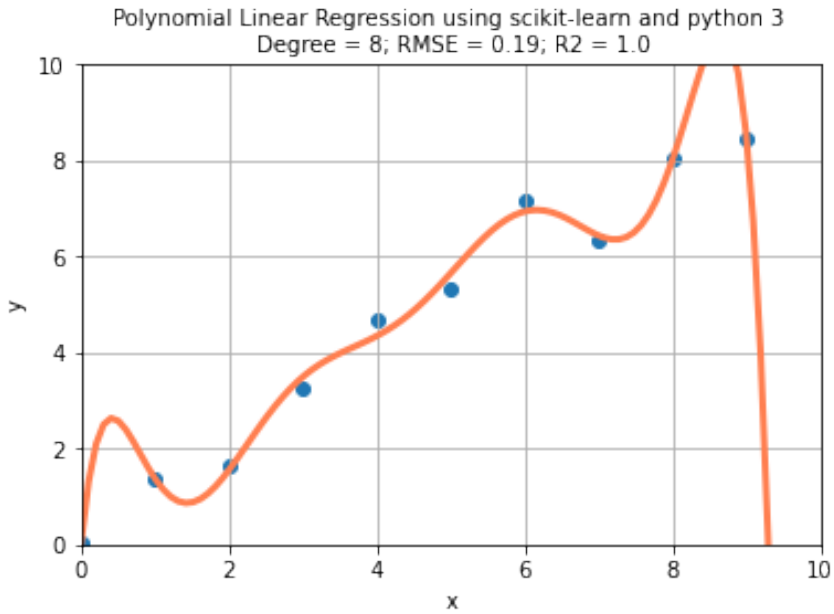
Hohe Komplexität

Optimierung auf geringen Trainingsfehler



$Err_{rmse} = 0.52$

Niedrige Komplexität



$Err_{rmse} = 0.19$

Hohe Komplexität

Frage: Welches ist das bessere Modell?

Expected Error, Verzerrung und Varianz

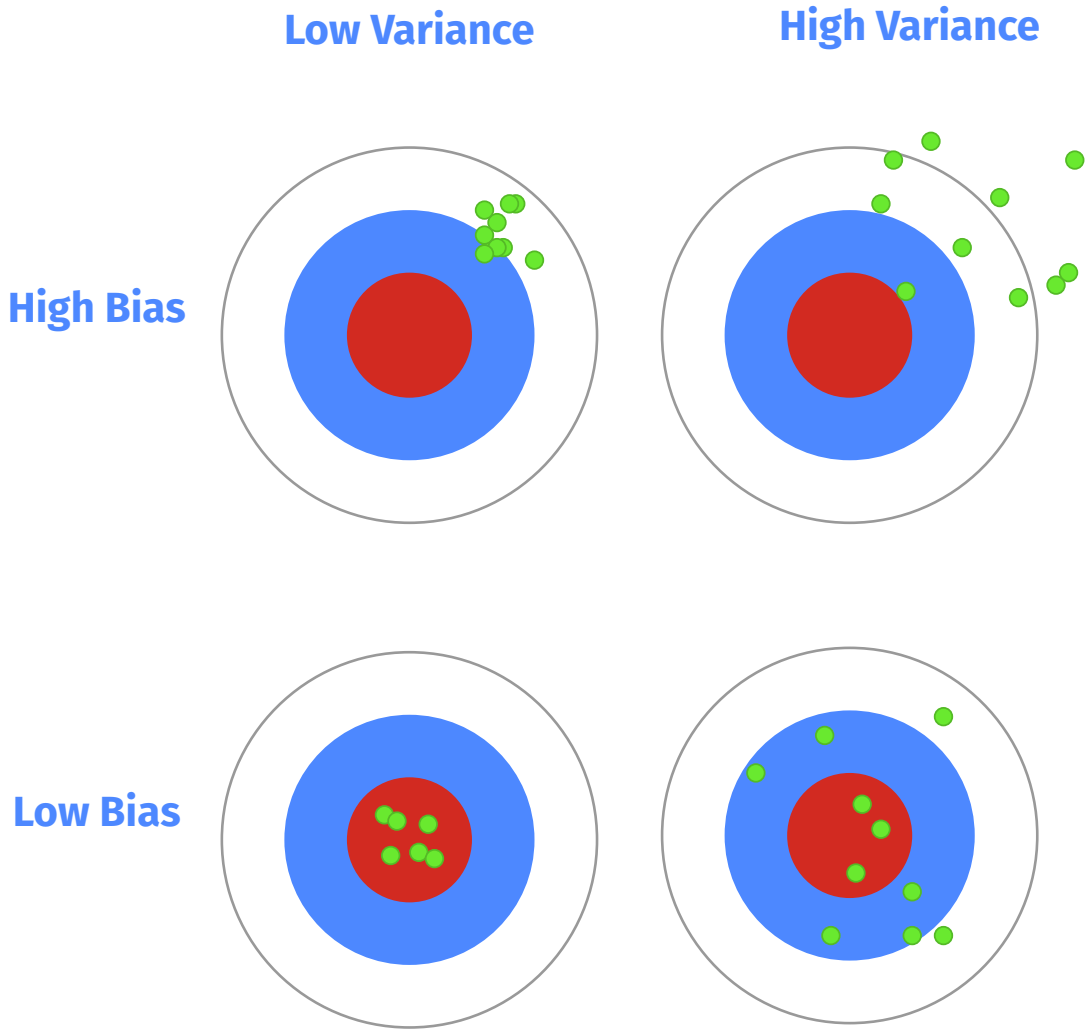
Der erwartete Fehler unseres Modells $\hat{f}$ ist

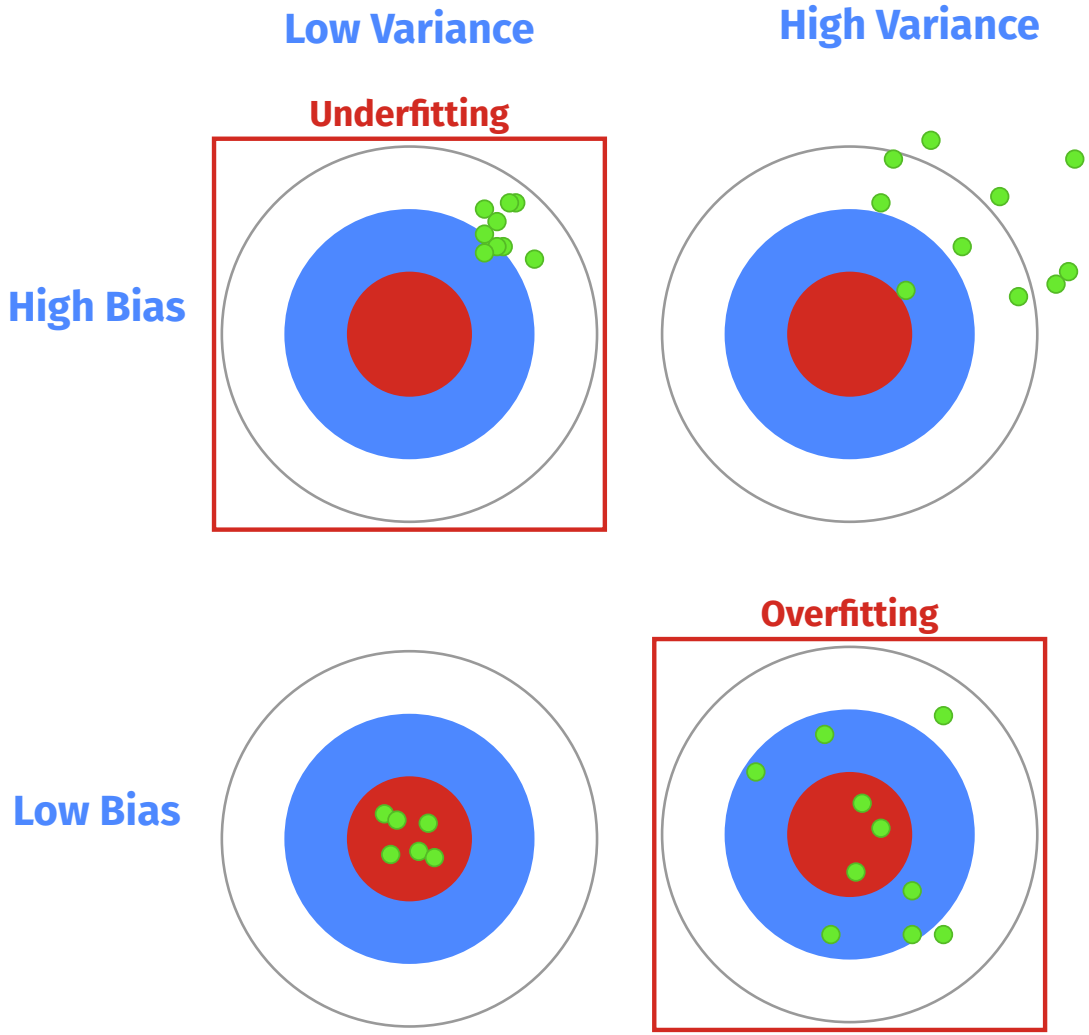
$$Err_{exp}(x) = E\left[(Y - \hat{f}(x))^2\right]$$

Err_{exp} läßt sich aufteilen in Verzerrung (Bias) und Varianz:

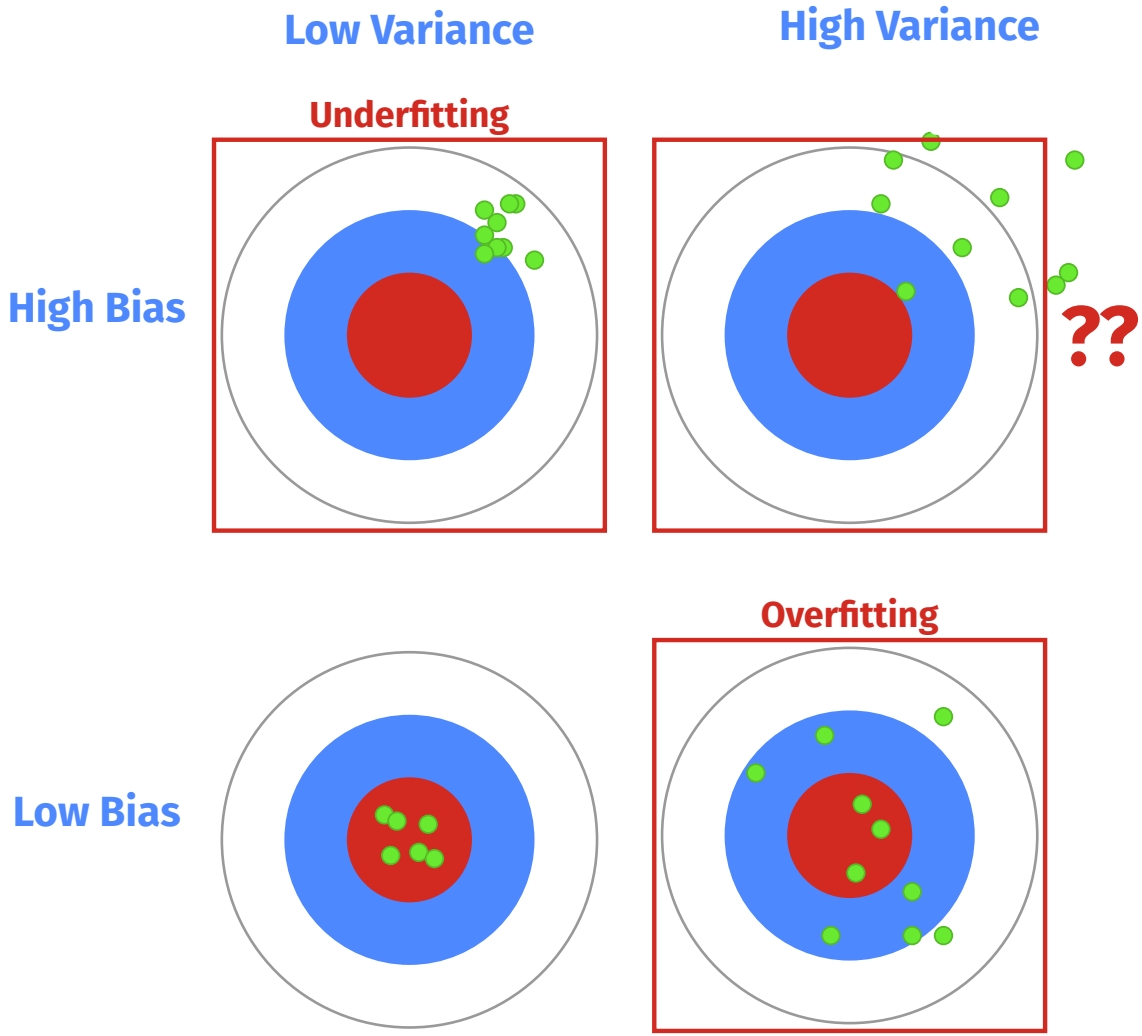
$$Err_{exp}(x) \quad = \quad \text{Bias}(\hat{f})^2 + \text{Var}(\hat{f}) + \text{unvermeidbarer Fehler}$$

Bias / Variance

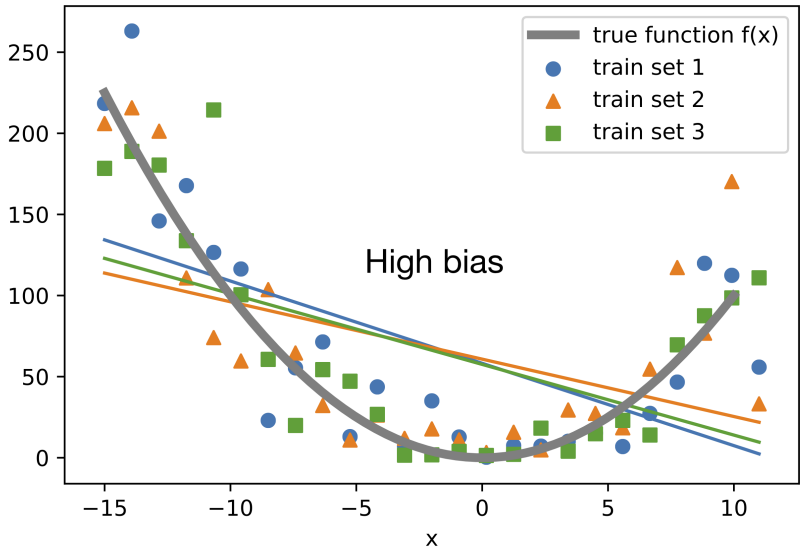


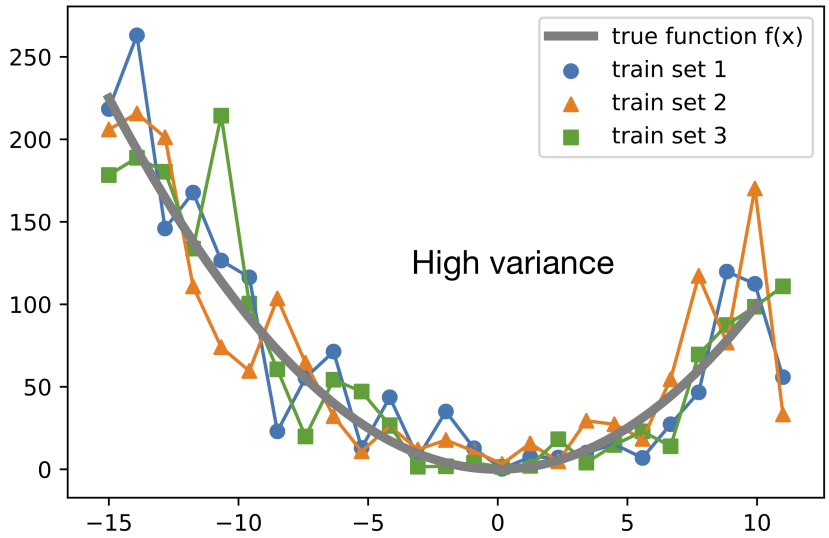

Low Variance
High Variance
Underfitting
High Bias
Low Bias
Overfitting

Bias / Variance

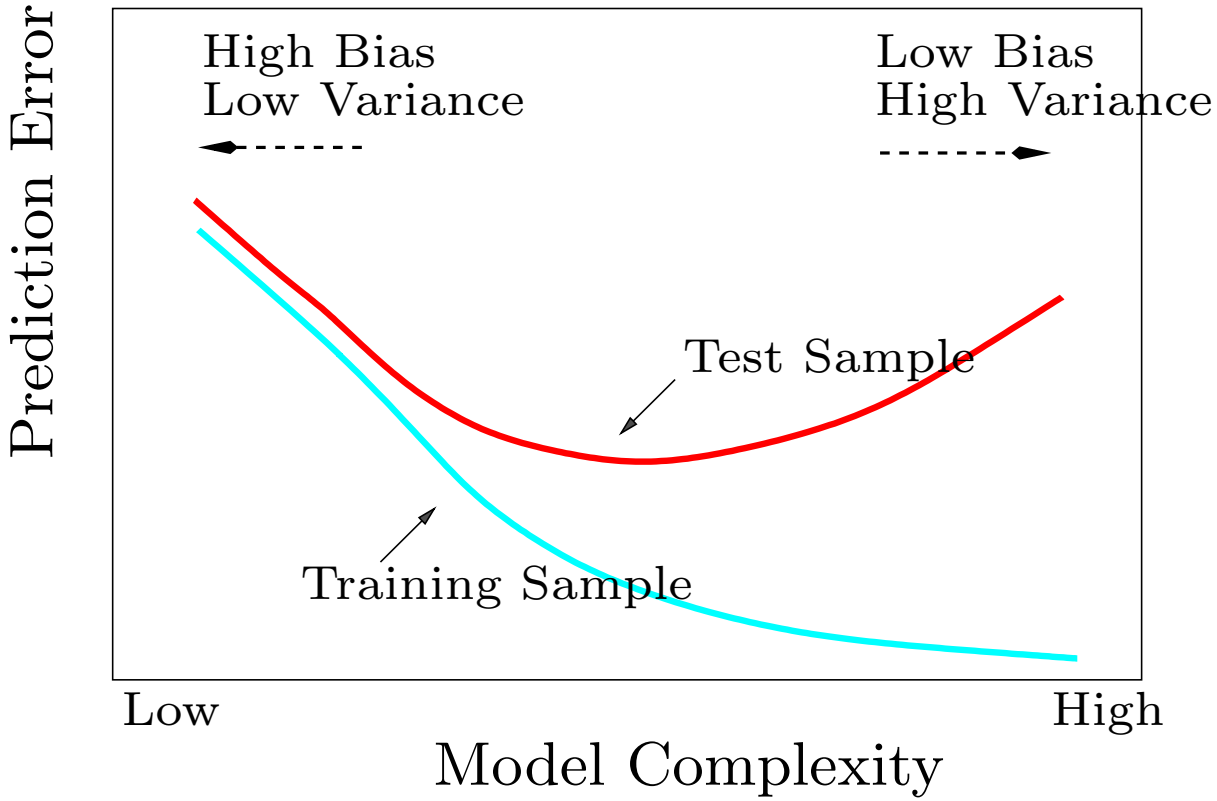


Beispiel: Bias / Variance





Bias / Variance Decomposition



Wie kontrollieren wir die Balance?

- Hohe Modell-Komplexität führt zu niedrigem Bias, hoher Varianz
- Niedrige Modell-Komplexität zu hohem Bias, niedriger Varianz

Modell-Komplexität

Modell-Komplexität ergibt sich z.B. aus

- dem Grad der Polynome (Regression)
- der Tiefe von Bäumen

Wie kontrollieren wir die Modell-Komplexität?

Maß für Modell-Komplexität in Fehler-Funktion aufnehmen:

$$\arg\min_{\mathbf{w}} \underbrace{\sum_{\mathbf{x},y} (y - f_{\mathbf{w}}(\mathbf{x}))^2}_{\text{Fehler}} + \lambda \underbrace{C(f_{\mathbf{w}})}_{\text{Komplexität}}$$

Parameter λ steuert Schwerpunkt (Fehler vs. Modellkomplexität)

Idee: Wenn w viele Nullen hat $\rightarrow$ geringe Komplexität

Wir können die Modellkomplexität an **w** "ablesen":

$$\mathbf{w}' = \begin{pmatrix} 4 \\ 0 \\ 1 \\ 8 \\ 31 \\ 19 \end{pmatrix} \qquad \mathbf{w}'' = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 19 \end{pmatrix}$$

$$4 + x^2 + 8x^3 + 31x^4 + 19x^5 \qquad 4x^2 + 19x^5$$

Idee: Wenn w viele Nullen hat $\rightarrow$ geringe Komplexität

Wir können die Modellkomplexität an **w** "ablesen":

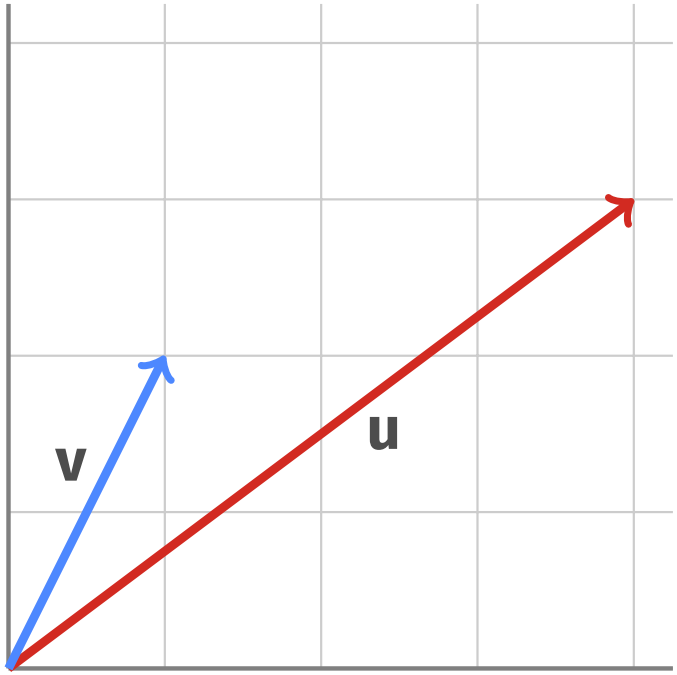
$$\mathbf{w}' = \begin{pmatrix} 4 \\ 0 \\ 1 \\ 8 \\ 31 \\ 19 \end{pmatrix} \qquad \mathbf{w}'' = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 19 \end{pmatrix}$$

$$4 + x^2 + 8x^3 + 31x^4 + 19x^5 \qquad 4x^2 + 19x^5$$

Statt Nullen sind auch viele kleine Werte gut, dann beeinflussen viele Komponenten die Funktion nicht so stark.

Frage: Wann hat ein Vektor w viele kleine Werte?

Norm eines Vektors **v** entspricht der Länge von **v**



$||v|| = 2.235$

$||u|| = 5.0$

Data Science 1: Foliensatz 7, Folie 9 und Übungsblatt 7

Regularisierung der Modellkomplexität

Modellkomplexität von $f_{\mathbf{w}}$ über Parametervektor $\mathbf{w}$ messen:

$$||\mathbf{w}||_2 = \sqrt{\sum_{i=1}^{p} w_i^2}$$

Regularisierung der Modellkomplexität

Modellkomplexität von $f_{\mathbf{w}}$ über Parametervektor $\mathbf{w}$ messen:

$$||\mathbf{w}||_2 = \sqrt{\sum_{i=1}^{p} w_i^2}$$

Beispiel: Ridge Regression

Optimierungsfunktion der Ridge Regression:

$$\arg\min_{\mathbf{w}} \left\{ \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda \sum_{i=1}^{p} w_i^2 \right\}$$

$||\mathbf{w}||_2^2$ als Komplexitätsmaß (quadrierte L_2-Norm)

SciKit Learn – Pre-Processing

Beispiel: Min-Max Skalierung

Alle numerischen Attribute werden auf das Interval $[0, 1]$ skaliert.

```
# Parameter bestimmen (fit):
x_min = min(df['x'])
x_max = max(df['x'])

# Daten transformieren (transform):
df['x'] = (df['x'] - x_min) / (x_max - x_min)
```

Data Science 1: Übungsblatt 6, Aufgabe 2 und Foliensatz 6, Folien 26+27

Beispiel: Min-Max Skalierung

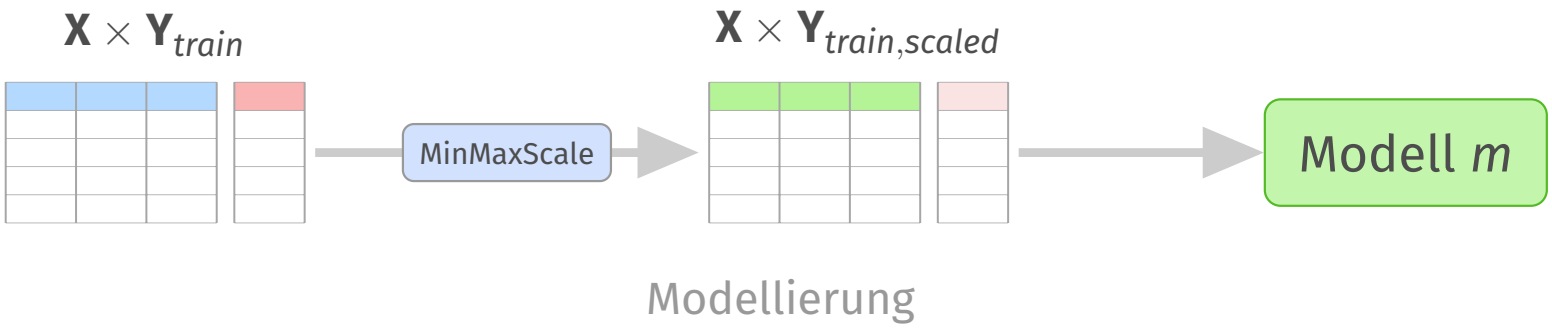
Alle numerischen Attribute werden auf das Interval $[0, 1]$ skaliert.

```
# Parameter bestimmen (fit):
x_min = min(df['x'])
x_max = max(df['x'])

# Daten transformieren (transform):
df['x'] = (df['x'] - x_min) / (x_max - x_min)
```

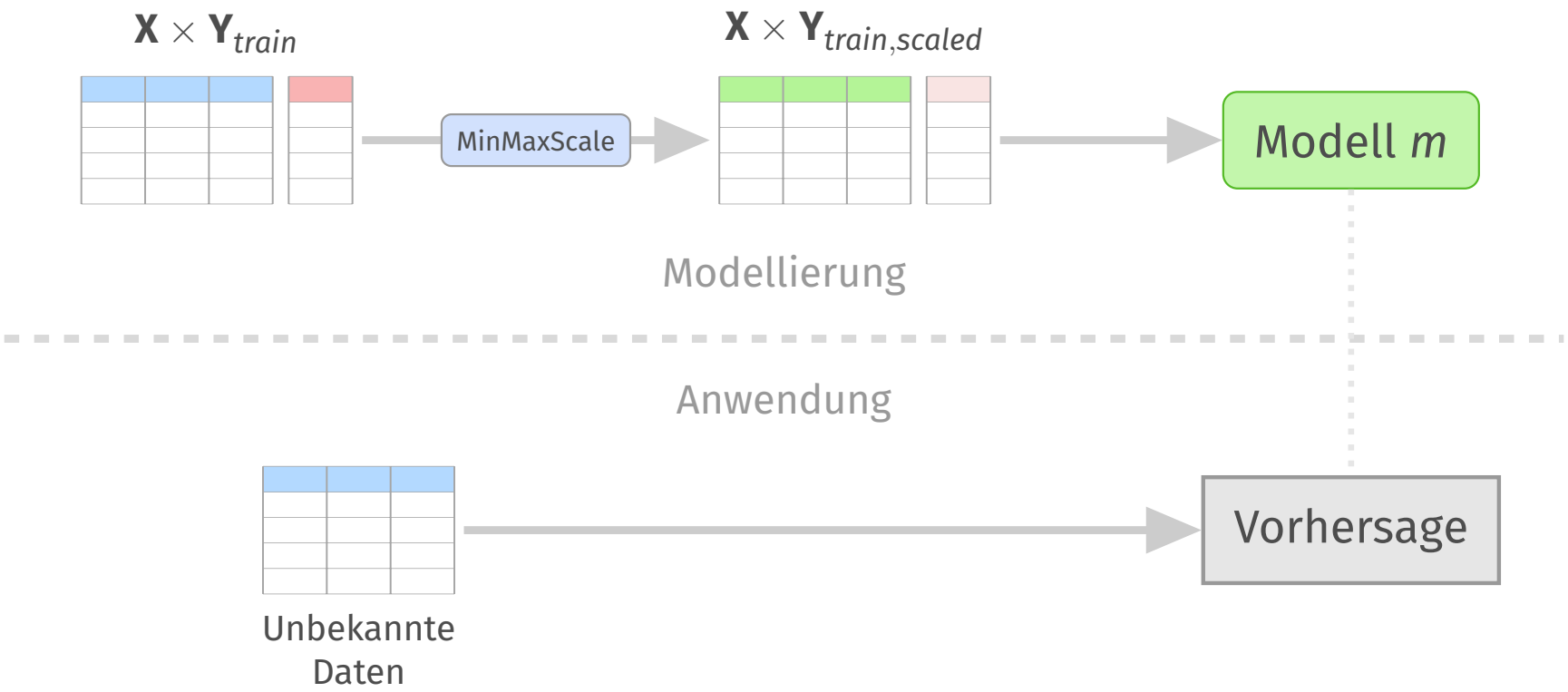
Data Science 1: Übungsblatt 6, Aufgabe 2 und Foliensatz 6, Folien 26+27

Die Parameter brauchen wir, wenn wir später Daten nochmal auf die gleiche Weise vorverarbeiten müssen!

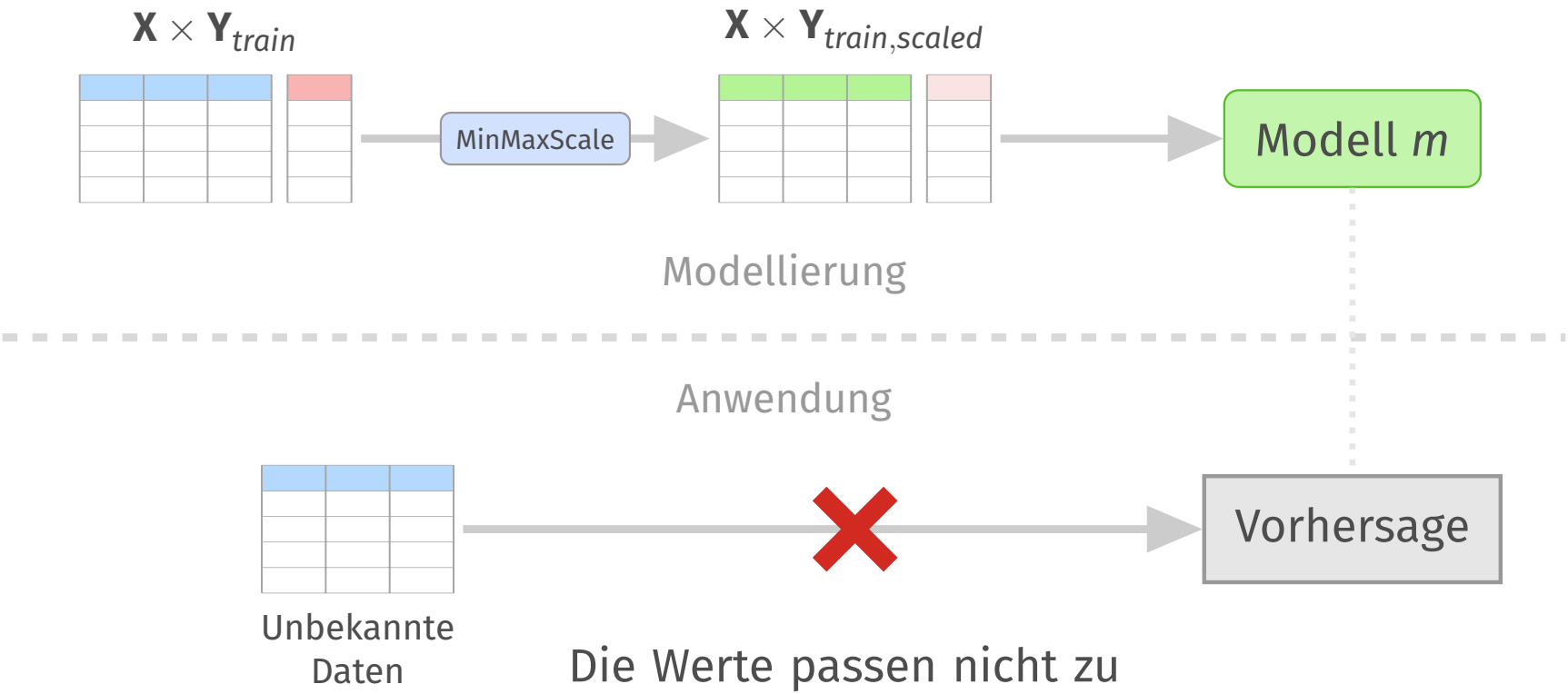
Beispiel: Min-Max Skalierung



Beispiel: Min-Max Skalierung

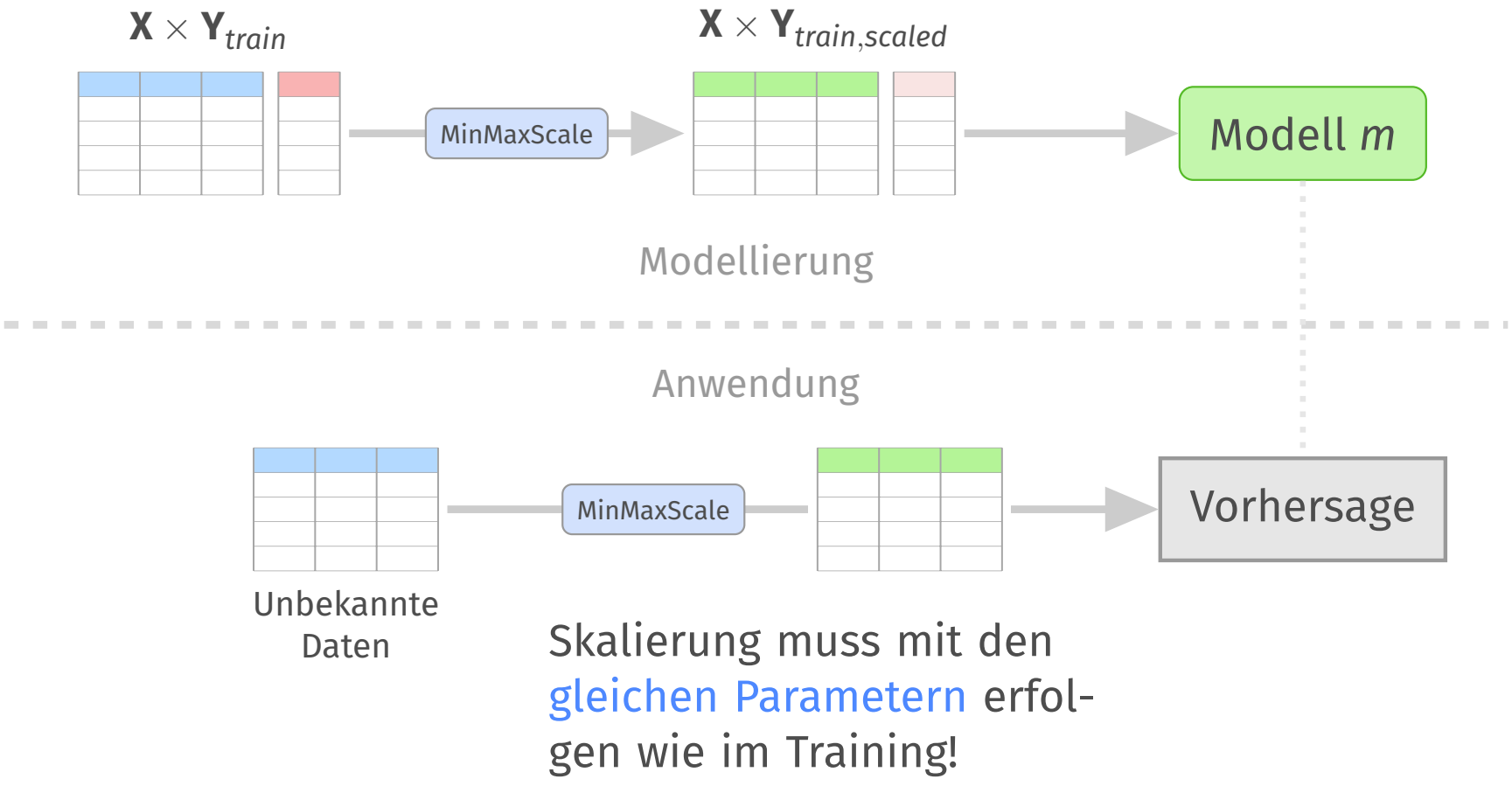


Beispiel: Min-Max Skalierung



Die Werte passen nicht zu den skalierten Daten, die das Modell gelernt hat.

Beispiel: Min-Max Skalierung

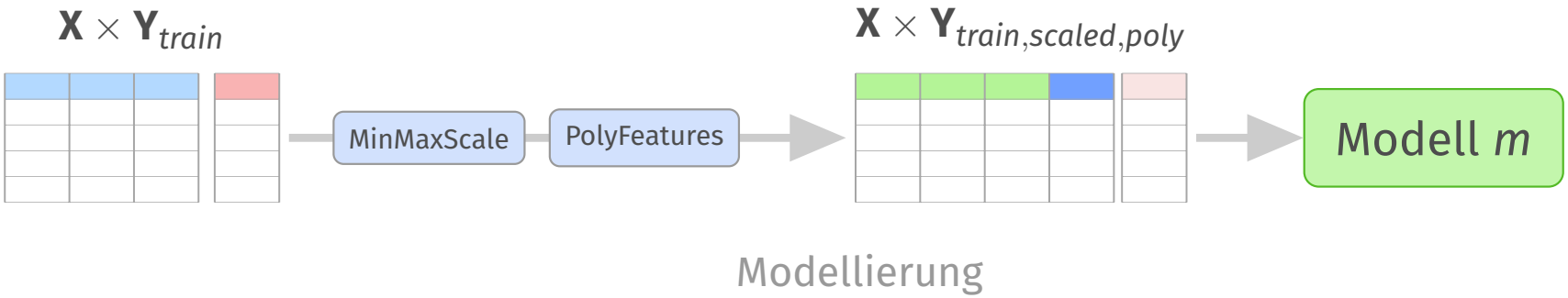


Skalierung muss mit den gleichen Parametern erfolgen wie im Training!

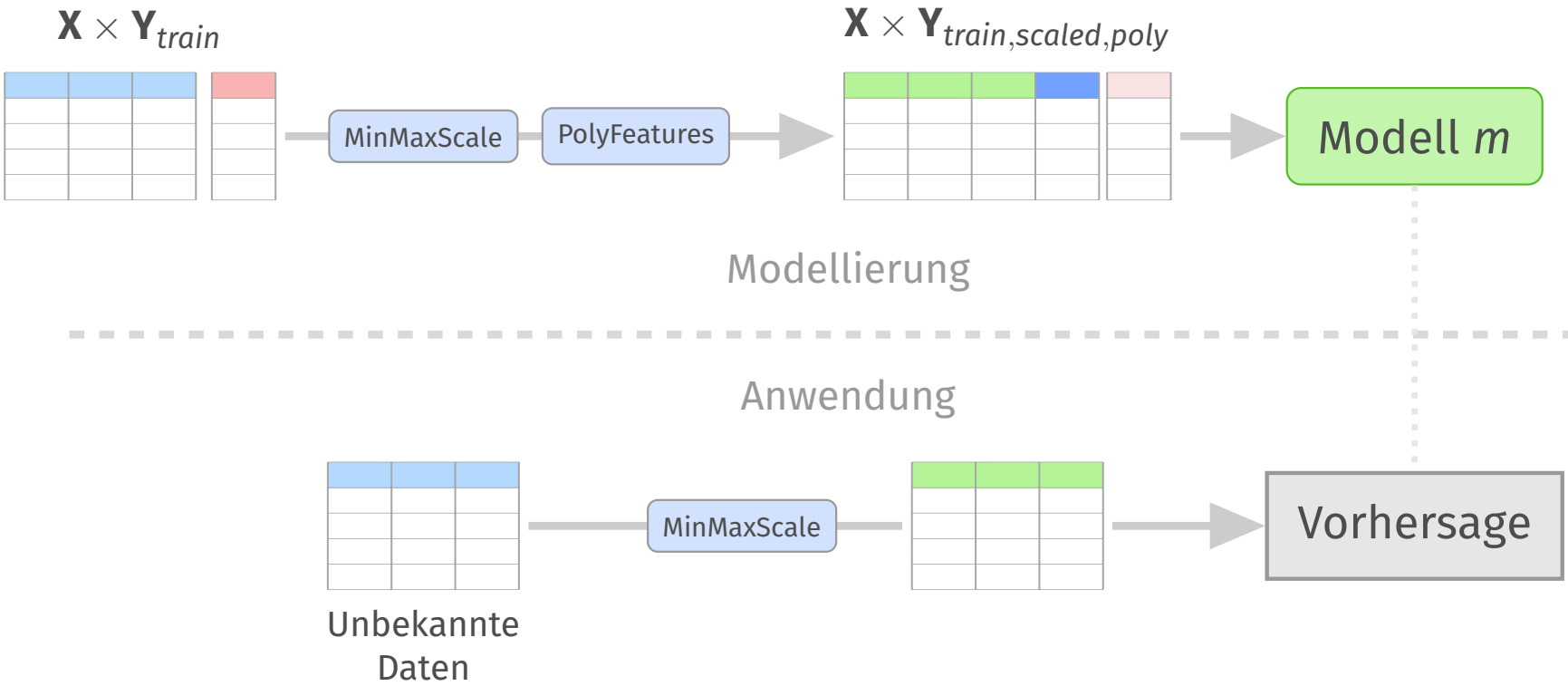
Min-Max Skalierung mit SciKit Learn

```python
from sklearn.preprocessing import MinMaxScaler

df = ... # read DataFrame

scaler = MinMaxScaler()
scaler.fit(df)

# scaler hat jetzt die min/max Werte gelernt:
scaler.data_max_
scaler.data_min_
```

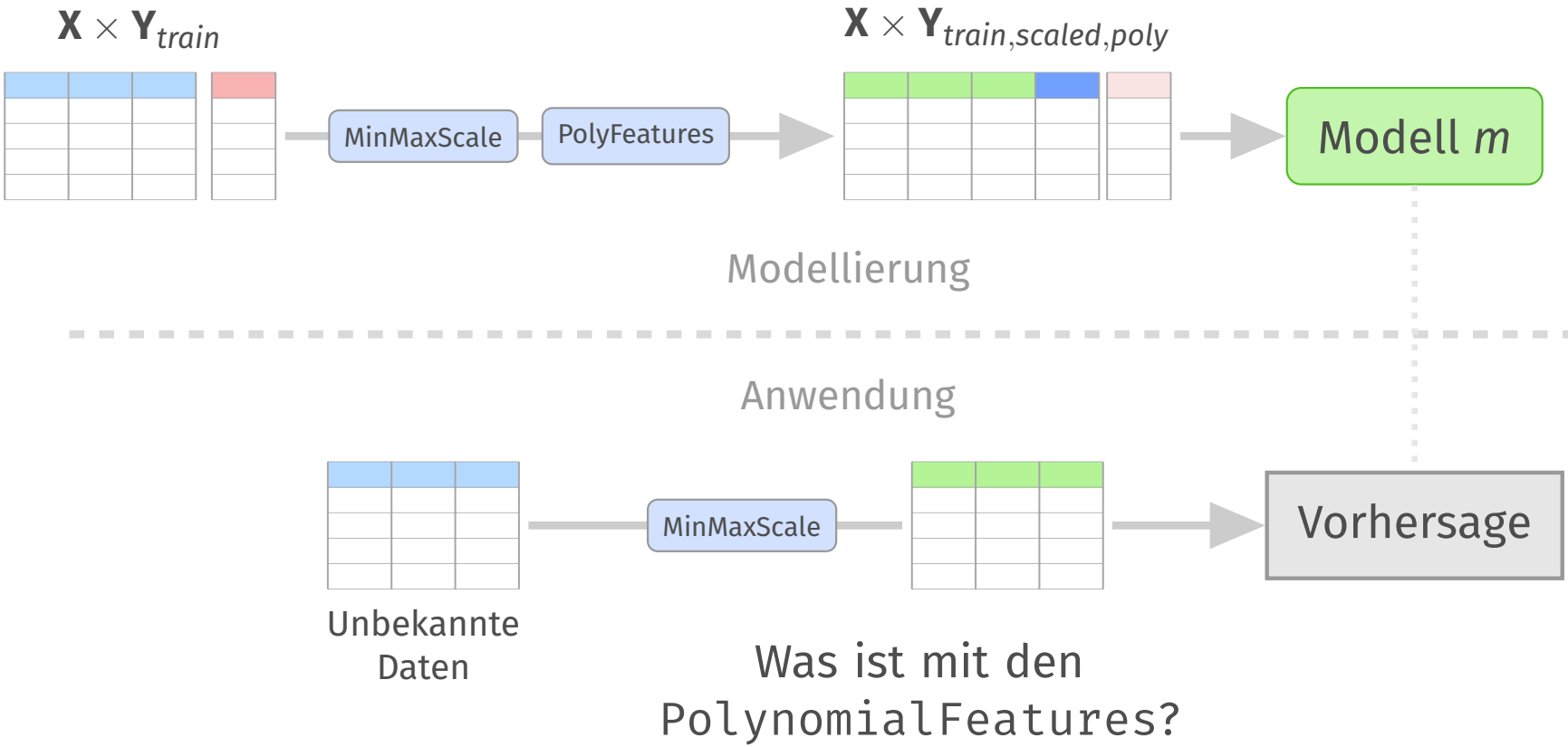
Beispiel: Min-Max Skalierung, Polynomial Regression



Beispiel: Min-Max Skalierung, Polynomial Regression



Beispiel: Min-Max Skalierung, Polynomial Regression



Pipeline: Mehrere Pre-Processing Schritte zusammengefasst

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

# Pipeline mit 2 Schritten:
scaler = MinMaxScaler()
poly = PolynomialFeatures(degree=4)

pipeline = Pipeline([('Scaling', scaler), ('Features'
                                 ,poly)])

# kurz:
pipeline = make_pipeline(scaler, poly)
```

Pipeline ist selbst wieder Preprocessor

Pipeline definieren und Schritte anpassen/anwenden:

```
# alle Schritte anpassen
pipeline.fit(data)

# alle Schritte anwenden
transformed = pipeline.transform(data)
```