# Data Science 2

## Lineare Regression

Prof. Dr. Christian Bockermann

Hochschule Bochum

Wintersemester 2021/2022

# INHALT

1 Regression

2 Lineare Regression

3 Fehlermaße

4 Multiple Lineare Regression

# Regression

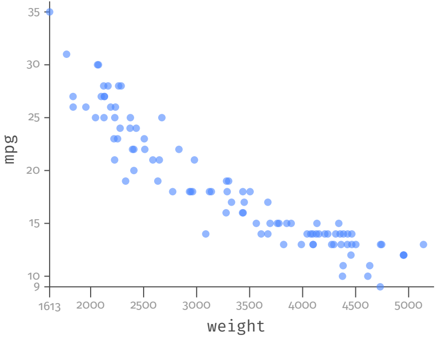
## Datensatz: Auto-MPG

| mpg | cylinders | displacement | horsepower | weight | acceleration | model year |
|---|---|---|---|---|---|---|
| 18 | 8 | 307 | 130 | 3504 | 12 | 70 |
| 15 | 8 | 350 | 165 | 3693 | 11.500 | 70 |
| 18 | 8 | 318 | 150 | 3436 | 11 | 70 |
| 16 | 8 | 304 | 150 | 3433 | 12 | 70 |
| 17 | 8 | 302 | 140 | 3449 | 10.500 | 70 |
| 15 | 8 | 429 | 198 | 4341 | 10 | 70 |

Daten von Autos inkl. Verbrauch:

- mpg - *Mileage per Gallon*
- Wieviele Meilen fährt das Auto mit 1 Gallone Sprit?
- 1 Gallone = 4.546 Liter

## Zusammenhang zwischen Gewicht und Verbrauch?



## Regression liefert reellwertige Vorhersagen

- Für Regression gilt $\mathcal{Y} = \mathbb{R}$
- Menge $\mathbf{X} \times \mathbf{y}$, d.h. jedem Beispiel $x_i$ ist ein $y_i \in \mathbb{R}$ zugeordnet
- Qualitätsfunktion $q : (\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \to \mathcal{Y}) \to \mathbb{R}$

**Ziel:**

- Finde Modell

$$f : \mathcal{X} \to \mathcal{Y},$$

das die Qualitätsfunktion <u>optimiert</u>.

**Regression liefert reellwertige Vorhersagen**

- Für Regression gilt $\mathcal{Y} = \mathbb{R}$
- Menge $\mathbf{X} \times \mathbf{y}$, d.h. jedem Beispiel $x_i$ ist ein $y_i \in \mathbb{R}$ zugeordnet
- Qualitätsfunktion $q : (\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \to \mathcal{Y}) \to \mathbb{R}$

**Ziel:**

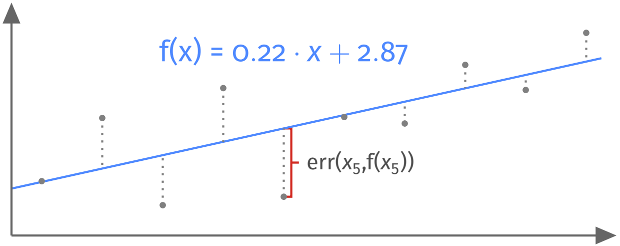
- Finde Modell

$$f : \mathcal{X} \to \mathcal{Y},$$

das die Qualitätsfunktion optimiert.

**Auch hier wieder: Lernen als Optimierungsproblem!**

## Beispiel: Regression

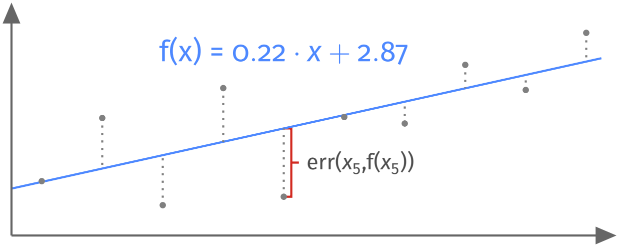



## Qualitätsfunktion:

Summe der Abstände von $f(x)$ zu den “richtigen” Werten

$$q(X,f) = \sum_{(x,y)\in X} (y - f(x))^2 = RSS(X,f)$$

## Beispiel: Regression




## Qualitätsfunktion:

Summe der Abstände von $f(x)$ zu den "richtigen" Werten

$$q(X,f) = \sum_{(x,y) \in X} (y - f(x))^2 = \boxed{RSS(X,f)}$$

**Residual Sum of Squares**

## Regression mit linearen Funktionen

Daten gegeben als Tabelle

| x | y |
|---|---|
| 1 | 3.105 |
| 2 | 4.147 |
| 3 | 2.709 |
| 4 | 4.640 |
| 5 | 2.848 |
| 6 | 4.163 |
| 7 | 4.056 |
| 8 | 5.023 |
| 9 | 4.609 |
| 10 | 5.551 |

Gesucht: lineare Funktion als Vorhersage Modell, z.B.

$$y = m \cdot x + b$$

**Einfache lineare Regression**

Problem ist Gleichungssystem mit $n$ Unbekannten:

$$y_i = m \cdot x_i + b$$

Schätzung der Regressionskoeffizienten:

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

$$b = \bar{y} - m \cdot \bar{x} \quad (2)$$

Vorheriges Beispiel in Matrix-Notation ergibt

$$\begin{pmatrix} 1 & 1 \\ 2 & 1 \\ \vdots & \vdots \\ 10 & 1 \end{pmatrix} \begin{pmatrix} m \\ b \end{pmatrix} = \begin{pmatrix} 3.105 \\ 4.147 \\ \vdots \\ 5.551 \end{pmatrix}$$

Dies führt zu folgender Formulierung des Optimierungsproblems:

$$\min_{\mathbf{x} \in \mathbb{R}^n} ||\mathbf{Ax} - \mathbf{y}||$$

## SciKit Learn enthält Lineare Regression

```
from sklearn.linear_model import LinearRegression

df = ...  # load dataframe
X = df['x']
y = df['y']

model = LinearRegression()
model.fit(X, y)
```

## SciKit Learn enthält Lineare Regression

```
from sklearn.linear_model import LinearRegression

df = ...  # load dataframe
X = df['x']
y = df['y']

model = LinearRegression()
model.fit(X, y)
```

**Führt zu Fehler: LinearRegression benötigt 2 Eingabe-Variablen!**

## Kurze Erinnerung:

Matrizen-Form

$$\begin{pmatrix} 1 & 1 \\ 2 & 1 \\ \vdots & \vdots \\ 10 & 1 \end{pmatrix} \begin{pmatrix} m \\ b \end{pmatrix} = \begin{pmatrix} 3.105 \\ 4.147 \\ \vdots \\ 5.551 \end{pmatrix}$$

DataFrame

| x |
|---|
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |

`df['x']`

| y |
|---|
| 3.105 |
| 4.147 |
| 2.709 |
| 4.640 |
| 2.848 |
| 4.163 |
| 4.056 |
| 5.023 |
| 4.609 |
| 5.551 |

`df['y']`

## Kurze Erinnerung:

### Matrizen-Form

$$\begin{pmatrix} 1 & 1 \\ 2 & 1 \\ \vdots & \vdots \\ 10 & 1 \end{pmatrix} \begin{pmatrix} m \\ b \end{pmatrix} = \begin{pmatrix} 3.105 \\ 4.147 \\ \vdots \\ 5.551 \end{pmatrix}$$

### DataFrame

| x |
|---|
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |

`df['x']`

| y |
|---|
| 3.105 |
| 4.147 |
| 2.709 |
| 4.640 |
| 2.848 |
| 4.163 |
| 4.056 |
| 5.023 |
| 4.609 |
| 5.551 |

`df['y']`

## Kurze Erinnerung:

Matrizen-Form

$$\begin{pmatrix} 1 & 1 \\ 2 & 1 \\ \vdots & \vdots \\ 10 & 1 \end{pmatrix} \begin{pmatrix} m \\ b \end{pmatrix} = \begin{pmatrix} 3.105 \\ 4.147 \\ \vdots \\ 5.551 \end{pmatrix}$$

DataFrame

| x | c |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |
| 6 | 1 |
| 7 | 1 |
| 8 | 1 |
| 9 | 1 |
| 10 | 1 |

| y |
|---|
| 3.105 |
| 4.147 |
| 2.709 |
| 4.640 |
| 2.848 |
| 4.163 |
| 4.056 |
| 5.023 |
| 4.609 |
| 5.551 |

`df[['x', 'c']]` `df['y']`

Spalte die *b* mit bestimmt

## SciKit Learn enthält Lineare Regression

```
from sklearn.linear_model import LinearRegression

df = ...  # load dataframe
df['c'] = 1
X = df['x', 'c']  # Konstante hinzufuegen fuer 'b'
y = df['y']

model = LinearRegression()
model.fit(X, y)

m = model.coef_[0]
b = model.intercept_
```

**Fehlermaße – Wie gut ist unser lineares Modell?**

Abstand zwischen Vorhersage und Wahrheit:

$$Err = \sum_{i=1}^{n} (\hat{y}_i - y_i)$$

**Fehlermaße – Wie gut ist unser lineares Modell?**

Abstand zwischen Vorhersage und Wahrheit:

$$Err = \sum_{i=1}^{n} (\hat{y}_i - y_i)$$

Was, wenn sich Fehler gegenseitig "korrigieren"?

Betrag liefert absolute error:

$$Err_{abs} = \sum_{i=1}^{n} |\hat{y}_i - y_i|$$

**Fehlermaße – Mean Squared Error**

Betrag ist unstetig – problematisch bei Optimierung, daher:

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

Quadrat verzerrt die Fehlereinschätzung etwas: `root mean squared error`

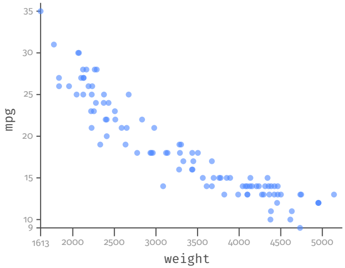
$$Err_{rmse} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2}$$

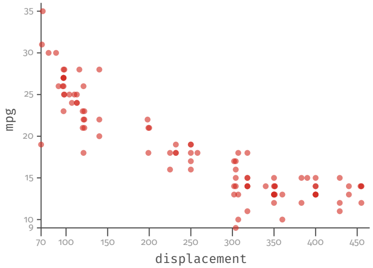
**SciKit Learn: `sklearn.metrics.mean_squared_error`**

```
from sklearn.metrics import mean_squared_error

yhat = model.predict(xs)
err_mse = mean_squared_error(y, yhat)

# root mean squared error:
err_rmse = mean_squared_error(y, yhat, squared=False)
```

# Multiple Lineare Regression

## Zusammenhang zwischen Gewicht, Hubraum und Verbrauch?



## Regression auf mehreren Variablen

Werden pro Datensatz p Merkmale gemessen, ergibt sich z.B.

$$\begin{aligned}
x_{1,1}m_1 + x_{1,2}m_2 + \ldots + x_{1,p}m_p &= y_1 \\
x_{2,1}m_1 + x_{2,2}m_2 + \ldots + x_{2,p}m_p &= y_2 \\
\vdots &= \vdots \\
x_{n,1}m_1 + x_{n,2}m_2 + \ldots + x_{n,p}m_p &= y_n
\end{aligned}$$

Dabei ist $x_{i,j}$ das $j$-te Merkmal des $i$-ten Datensatzes

## Regression auf mehreren Variablen – Kundendaten

Werden pro Datensatz p Merkmale gemessen, ergibt sich z.B.

$$
\begin{aligned}
k_{1,discount} m_1 + k_{1,sport} m_2 + \ldots + k_{1,luxus} m_p &= y_1 \\
k_{2,discount} m_1 + k_{2,sport} m_2 + \ldots + k_{2,luxus} m_p &= y_2 \\
\vdots &= \vdots \\
k_{n,discount} m_1 + k_{n,sport} m_2 + \ldots + k_{n,luxus} m_p &= y_n
\end{aligned}
$$

Dabei ist $k_{i,j}$ das $j$-Eigenschaft von Kunde $i$, z.B. die Discount-Affinität, Vorliebe für Sport, Marke Luxus, usw.

## Regression auf mehreren Variablen

| discount | sport | fashion | beauty |
|---|---|---|---|
| 0.683 | 0.130 | 0.370 | 0.350 |
| 0.107 | 0.080 | 0.300 | 0.410 |
| 0.419 | 0.060 | 0.240 | 0.510 |
| 0.119 | 0.130 | 0.320 | 0.360 |
| 0.825 | 0.160 | 0.340 | 0.350 |

$$\cdot \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \end{bmatrix} =$$

| sales |
|---|
| 2290 |
| 2782 |
| 7869 |
| 3976 |
| 3598 |

$$\mathbf{A} \qquad \mathbf{x} = \mathbf{y}$$

## Regression auf mehreren Variablen

| discount | sport | fashion | beauty |
|---|---|---|---|
| 0.683 | 0.130 | 0.370 | 0.350 |
| 0.107 | 0.080 | 0.300 | 0.410 |
| 0.419 | 0.060 | 0.240 | 0.510 |
| 0.119 | 0.130 | 0.320 | 0.360 |
| 0.825 | 0.160 | 0.340 | 0.350 |

$\cdot$

| |
|---|
| $m_1$ |
| $m_2$ |
| $m_3$ |
| $m_4$ |

=

| sales |
|---|
| 2290 |
| 2782 |
| 7869 |
| 3976 |
| 3598 |

$$\mathbf{A} \quad \mathbf{x} = \mathbf{y}$$

## Regression auf mehreren Variablen

| discount | sport | fashion | beauty |
|---|---|---|---|
| 0.683 | 0.130 | 0.370 | 0.350 |
| 0.107 | 0.080 | 0.300 | 0.410 |
| 0.419 | 0.060 | 0.240 | 0.510 |
| 0.119 | 0.130 | 0.320 | 0.360 |
| 0.825 | 0.160 | 0.340 | 0.350 |

$$\cdot \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \end{bmatrix} =$$

| sales |
|---|
| 2290 |
| 2782 |
| 7869 |
| 3976 |
| 3598 |

$$\mathbf{A} \quad \mathbf{x} = \mathbf{y}$$

## Regression auf mehreren Variablen

| discount | sport | fashion | beauty |
|---|---|---|---|
| 0.683 | 0.130 | 0.370 | 0.350 |
| 0.107 | 0.080 | 0.300 | 0.410 |
| 0.419 | 0.060 | 0.240 | 0.510 |
| 0.119 | 0.130 | 0.320 | 0.360 |
| 0.825 | 0.160 | 0.340 | 0.350 |

$\cdot$

| |
|---|
| $m_1$ |
| $m_2$ |
| $m_3$ |
| $m_4$ |

$=$

| sales |
|---|
| 2290 |
| 2782 |
| 7869 |
| 3976 |
| 3598 |

$$\mathbf{A} \quad \mathbf{x} = \mathbf{y}$$

## Regression auf mehreren Variablen

| discount | sport | fashion | beauty |
|---|---|---|---|
| 0.683 | 0.130 | 0.370 | 0.350 |
| 0.107 | 0.080 | 0.300 | 0.410 |
| 0.419 | 0.060 | 0.240 | 0.510 |
| 0.119 | 0.130 | 0.320 | 0.360 |
| 0.825 | 0.160 | 0.340 | 0.350 |

$$\cdot \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \end{bmatrix} =$$

| sales |
|---|
| 2290 |
| 2782 |
| 7869 |
| 3976 |
| 3598 |

$$\mathbf{A} \quad \mathbf{x} = \mathbf{y}$$

Ürsprüngliches Problem

$$\mathbf{A}\mathbf{x} = \mathbf{y}$$

$\mathbf{A}$ und $\mathbf{y}$ als Trainingsdaten vorhanden

$$\begin{aligned} & \mathbf{A}\mathbf{x} &&= \mathbf{y} \\ \Leftrightarrow\quad & \mathbf{A}^T\mathbf{A}\mathbf{x} &&= \mathbf{A}^T\mathbf{y} \\ \Leftrightarrow\quad & \mathbf{x} &&= (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y} \end{aligned}$$

Analytisch lösbar, falls $\mathbf{A}^T\mathbf{A}$ invertierbar.
Verschiedene Approximationsverfahren bei nicht-Invertierbarkeit

## SciKit Learn unterstützt Multi-Regression

```
df = pd.read_csv("data/auto-mpg.csv")

X = df[['weight', 'displacement', 'cylinders', '
                              acceleration']]
y = df['mpg']

model = LinearRegression()
model.fit(X, y)

yhat = model.predict(X)
err_mse = mean_squared_error(y, yhat)
```

◀ Notebook: Kurs/DataScience2/V2-1-LinReg

**Trainings- und Test-Fehler betrachten!**

- Gezeigte Schritte berechnen den Test-Fehler
- Train/Test-Split wie bei Klassifikation!
- Auch Regression kann in Overfitting enden!