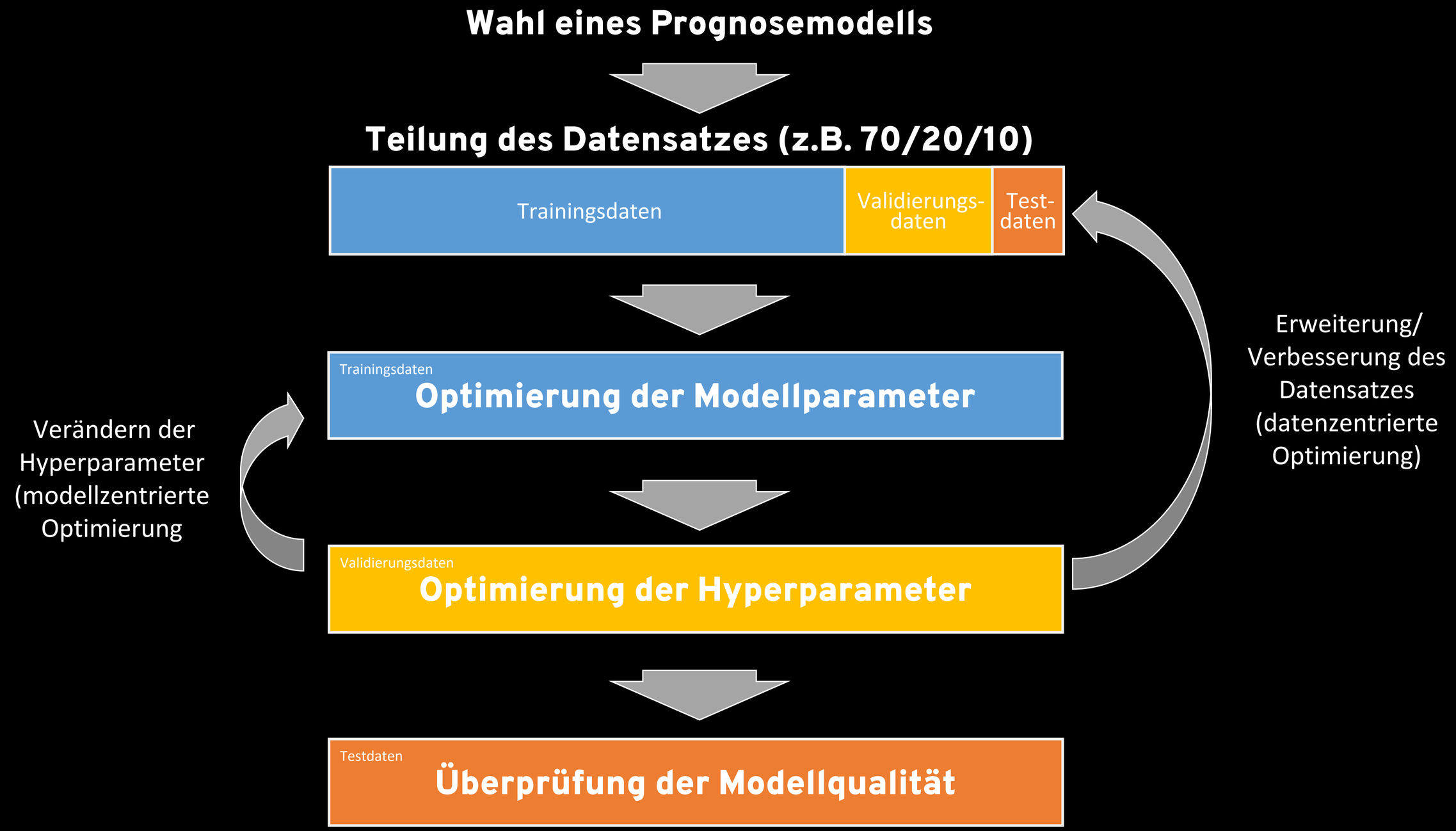
22.11.22

OPENCAMPUS.sh

# Einführung in Data Science und maschinelles Lernen

# EINFÜHRUNG IN MASCHINELLES LERNEN

- Vorgehen zum Trainieren von maschinellen Lernalgorithmen
- Definition der linearen Regression
- Kostenfunktionen
- Optimierungsfunktionen
- Overfitting
- Regularisierung


Wahl eines Prognosemodells
Teilung des Datensatzes (z.B. 70/20/10)
Trainingsdaten
Validierungs-daten
Test-daten
Trainingsdaten
Optimierung der Modellparameter
Validierungsdaten
Optimierung der Hyperparameter
Testdaten
Überprüfung der Modellqualität
Verändern der Hyperparameter (modellzentrierte Optimierung
Erweiterung/ Verbesserung des Datensatzes (datenzentrierte Optimierung)

# PROGNOSEMODELLE

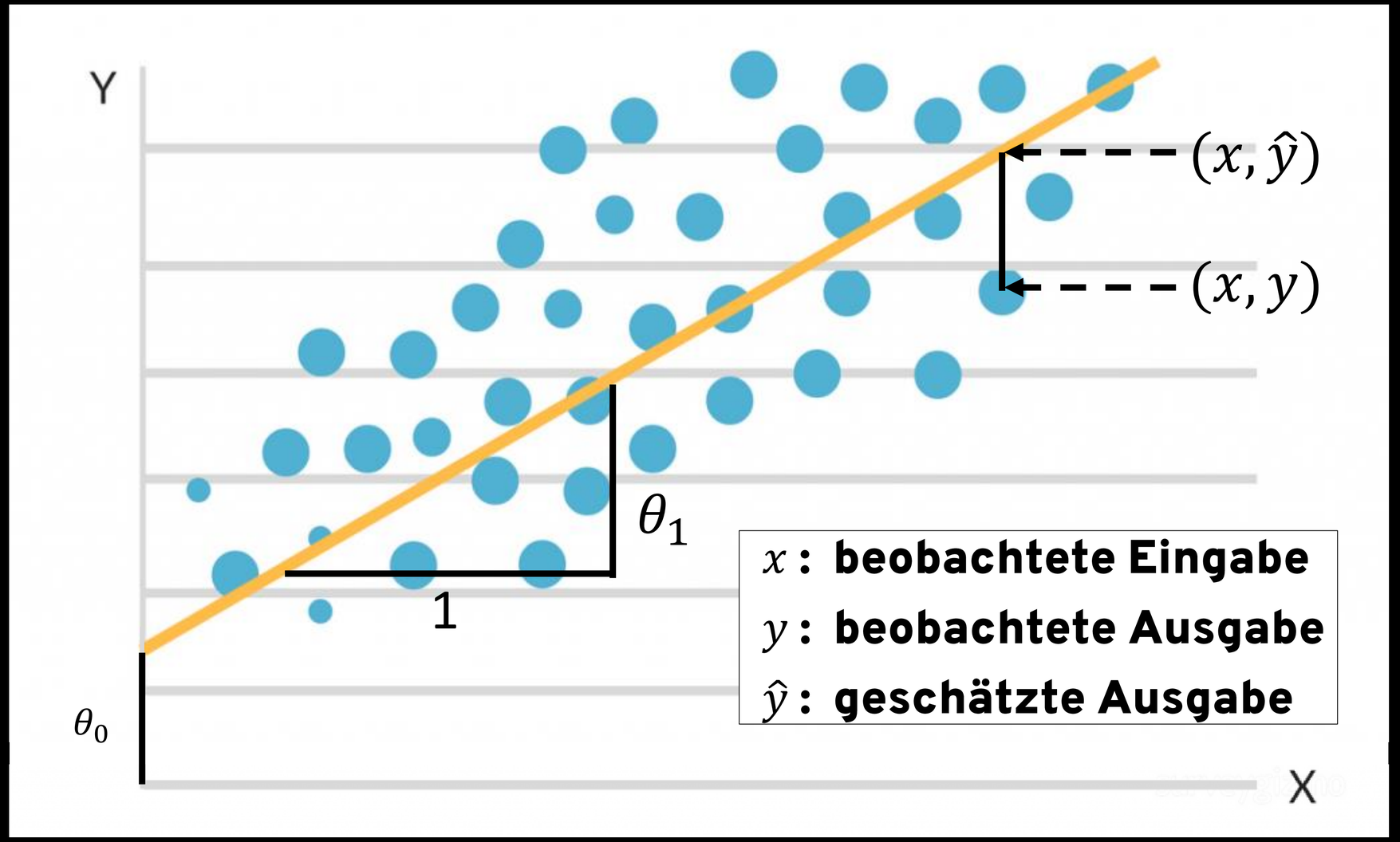
**In diesem Kurs behandelte:**

- **Lineares Modell**
- **Neuronales Netz**
- **(Support Vektor Maschine)**

# LINEARES MODELL

$$\hat{y} = \theta_0 + \theta_1 x$$

$$= h_x\,(\theta_0, \theta_1)$$



# KOSTENFUNKTION

**Zur Berechnung der Funktion mit den optimalen Parametern $\theta_0$ und $\theta_1$:**

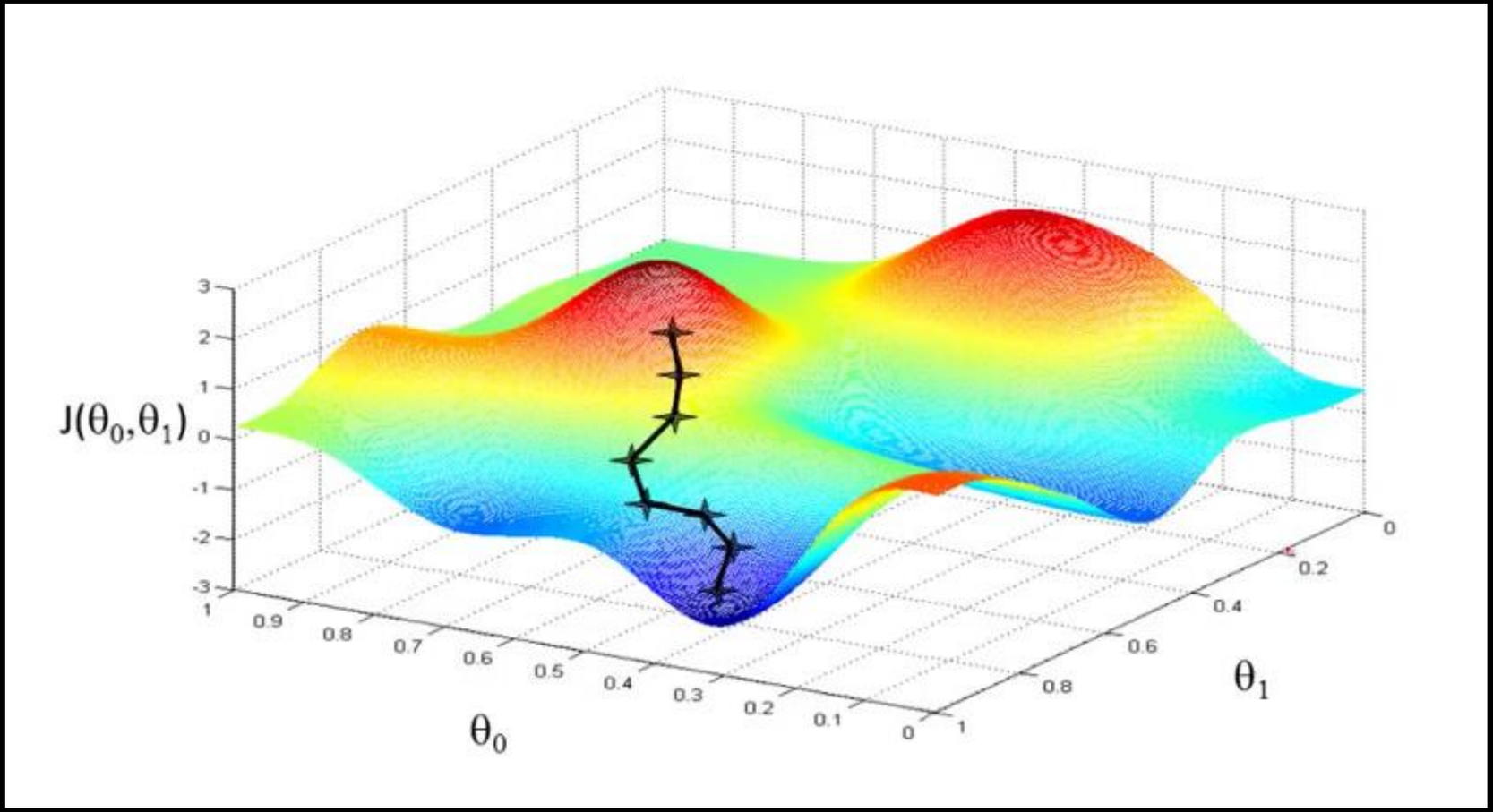
$$J_x(\theta_0, \theta_1) = \qquad h_x(\theta_0, \theta_1) - y$$

**Mean Absolute Error (MAE)**

$$J_x(\theta_0, \theta_1) = \frac{1}{m} \sum (h_x(\theta_0, \theta_1) - y)^2$$

**Mean Squared Error (MSE)**

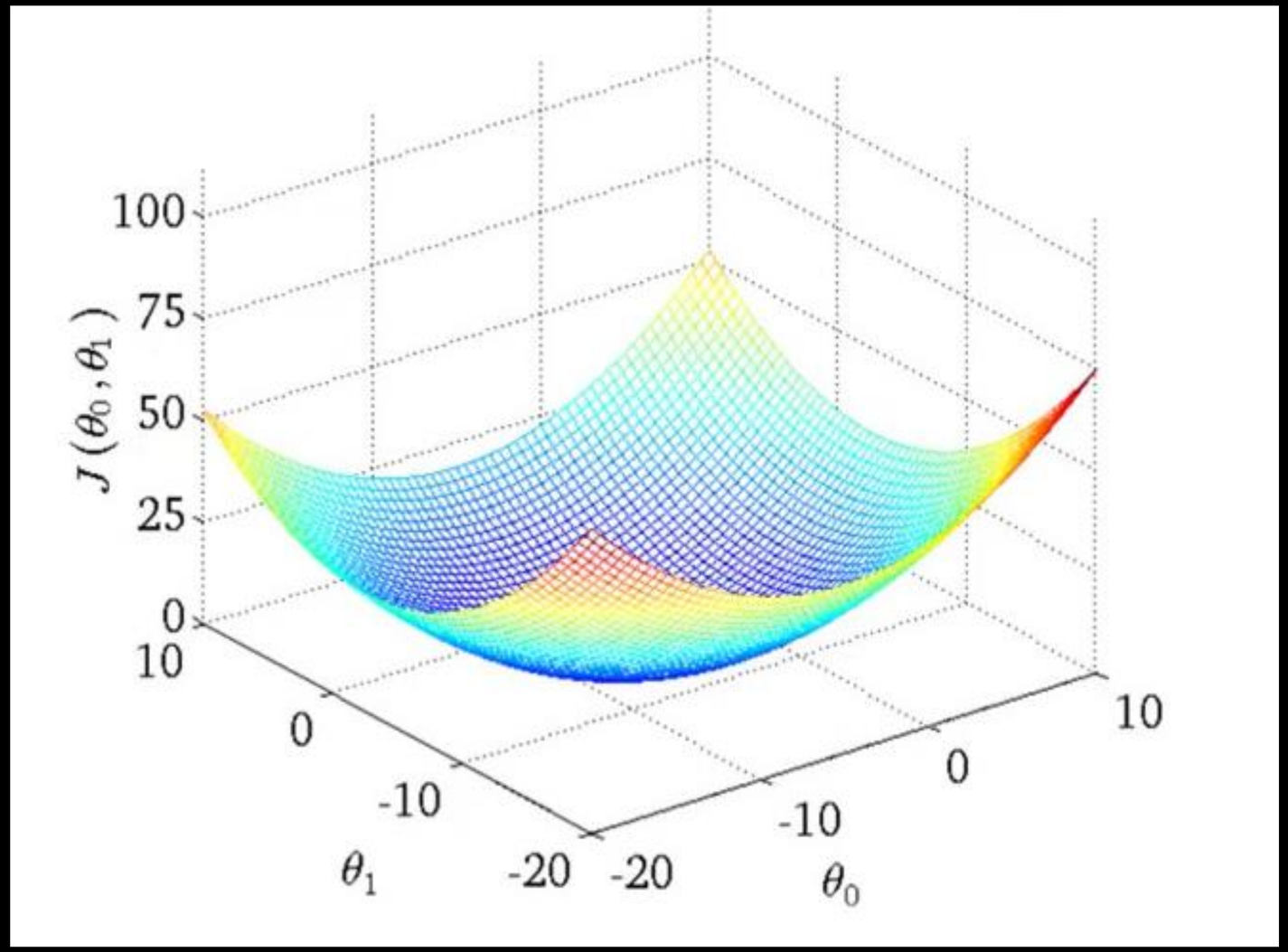
# MINIMIERUNG DER KOSTENFUNKTION




- Über ein iteratives Verfahren (Gradient Descent), das sich dem Minimum annähert
- Schrittgröße zur Annäherung wird durch die Lernrate („Learning Parameter“) kontrolliert

Quelle: https://www.coursera.org/learn/machine-learning

# MINIMIERUNG BEI LINEAREN MODELLEN




Quelle: https://www.coursera.org/learn/machine-learning

- **Für lineare Modelle ist die Kostenfunktion konvex und besitzt keine lokalen Minima.**
- **Hier werden häufig auch andere statistische Verfahren als Gradient Descent genutzt.**

  **(Insbesondere wenn das Modell nur wenige Variablen umfasst.)**

# BEISPIEL EINES LINEAREN MODELLS

```
library(readr)
house_pricing <- read_csv("https://raw.githubusercontent.com/
                          opencampus-sh/einfuehrung-in-data-
                          science-und-ml/main/house_pricing_data/
                          house_pricing_train.csv")


mod <- lm(price ~ sqft_lot15 + as.factor(condition), house_pricing)
summary(mod)
```

# ERGEBNIS DES LINEAREN MODELLS

```
Call:
lm(formula = price ~ sqft_lot15 + as.factor(condition), data = house_pricing)

Residuals:
    Min      1Q  Median      3Q     Max
-795388 -214555  -85989  101761 7183941

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)              3.261e+05  7.049e+04   4.625 3.77e-06 ***
sqft_lot15               1.138e+00  1.035e-01  11.002  < 2e-16 ***
as.factor(condition)2   -2.305e+04  7.690e+04  -0.300 0.764379
as.factor(condition)3    2.031e+05  7.057e+04   2.878 0.004008 **
as.factor(condition)4    1.800e+05  7.070e+04   2.546 0.010915 *
as.factor(condition)5    2.762e+05  7.118e+04   3.880 0.000105 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 366300 on 17284 degrees of freedom
Multiple R-squared:  0.01409,   Adjusted R-squared:  0.0138
F-statistic: 49.39 on 5 and 17284 DF,  p-value: < 2.2e-16
```

# KENNWERTE DER REGRESSION

## p-Wert

- **Signifikanzwert**
- **Wahrscheinlichkeit, dass der zugehörige Wert in der Regression gleich null ist.**
- **Werte unter 0,05 (entspricht 5%) werden üblicherweise als signifikant betrachtet**

## (Adjustiertes) $R^2$

- **Kennzahl zur Beurteilung der Güte einer Regression**
- **Wert zwischen 0 und 1, der der dem Anteil der erklärten Variation entspricht (1 entspricht 100%):**

$$R^2 = \frac{\text{Erklärte Varianz}}{\text{Gesamtvarianz}}$$

- **Das adjustierte $R^2$ bestraft das Hinzufügen zusätzlicher Variablen/Parameter**

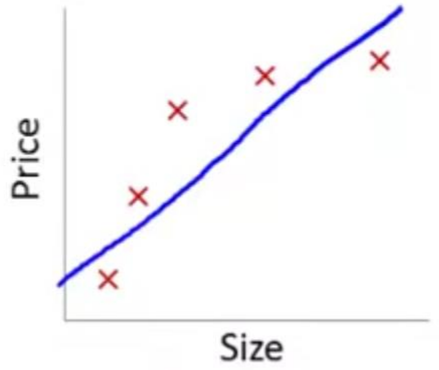
DataCamp Tutorial zur Linearen Modellierung in R :
https://www.datacamp.com/community/tutorials/linear-regression-R

# BREAKOUT

- Stellt für Euren Datensatz eine Modellgleichung auf und berechnet das adjustierte $R^2$ auf Basis eines linearen Modells.

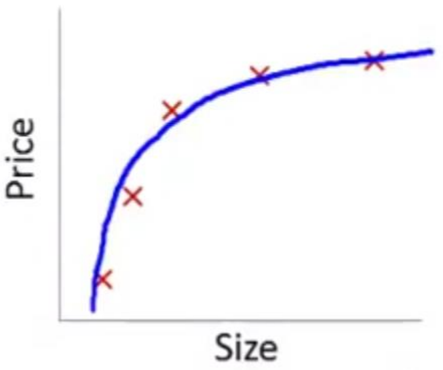
# OVERFITTING

Example: Linear regression (housing prices)



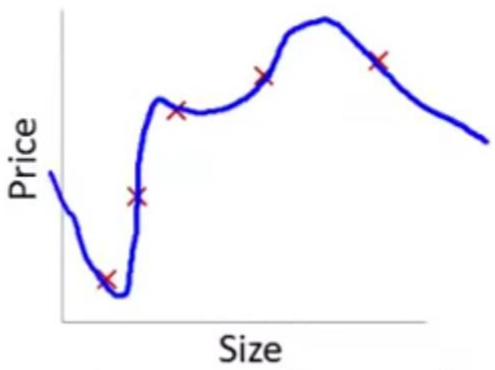

$\rightarrow \theta_0 + \theta_1 x$

"Underfit" "High bias"




$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2$

"Just right"




$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

"Overfit" "High variance"

**Overfitting:** If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples).

Computer Science Engineering. (2017). *3.4.1 The Problem of Overfitting by Andrew Ng*. Abgerufen von https://www.youtube.com/watch?v=OSd30QGMl88

# BEISPIEL ZU OVERFITTING

````
---
title: "Linear Regression"
output: html_notebook
---

```{r}
# Importing Function Packages
library(dplyr)
library(readr)
library(lubridate)
library(broom)
library(Metrics)
```


```{r}
# Importing Training and Test Data
house_pricing_train <- read_csv("./house_pricing_data/house_pricing_train.csv")
house_pricing_test <- read_csv("./house_pricing_data/house_pricing_test.csv")
```


```{r}
# Estimating (Training) Models
mod1 <- lm(price ~ bathrooms, house_pricing_train)
mod2 <- lm(price ~ as.factor(bathrooms), house_pricing_train)
mod3 <- lm(price ~ as.factor(bathrooms) + as.factor(zipcode), house_pricing_train)
mod4 <- lm(price ~ as.factor(bathrooms) + as.factor(zipcode) + condition, house_pricing_train)
mod5 <- lm(price ~ as.factor(bathrooms) + as.factor(zipcode) + as.factor(condition), house_pricing_train)
mod6 <- lm(price ~ as.factor(bathrooms) + as.factor(zipcode) + as.factor(condition) + sqft_living15, house_pricing_train)
mod7 <- lm(price ~ as.factor(bathrooms) + as.factor(zipcode) + as.factor(condition) + sqft_living15 + floors + view + grade +
as.factor(zipcode)*as.factor(bathrooms), house_pricing_train)
```


```{r}
summary(mod1)
````
````

# STRATEGIEN ZUR VERMEIDUNG VON OVERFITTING

Options:

1. Reduce number of features.
   - — Manually select which features to keep.
   - — Model selection algorithm

2. Regularization.
   - — Keep all the features, but reduce magnitude/values of parameters $\theta_j$.
   - — Works well when we have a lot of features, each of which contributes a bit to predicting $y$.

Computer Science Engineering. (2017). *3.4.1 The Problem of Overfitting by Andrew Ng*. Abgerufen von https://www.youtube.com/watch?v=OSd30QGMl88

# REGULARISIERUNG

**„Bestrafen“ der Verwendung von Variableninformation im Rahmen der Kostenfunktion**

**Lineares Modell mit mehreren Variablen $x_1, x_2$ und vielen möglichen weiteren:**

$$h_x(\theta_0, \theta_1, \theta_2, \dots) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots$$

**(mit $\theta_0, \theta_1, \theta_2, \dots$ als den zu schätzenden Modellparametern)**

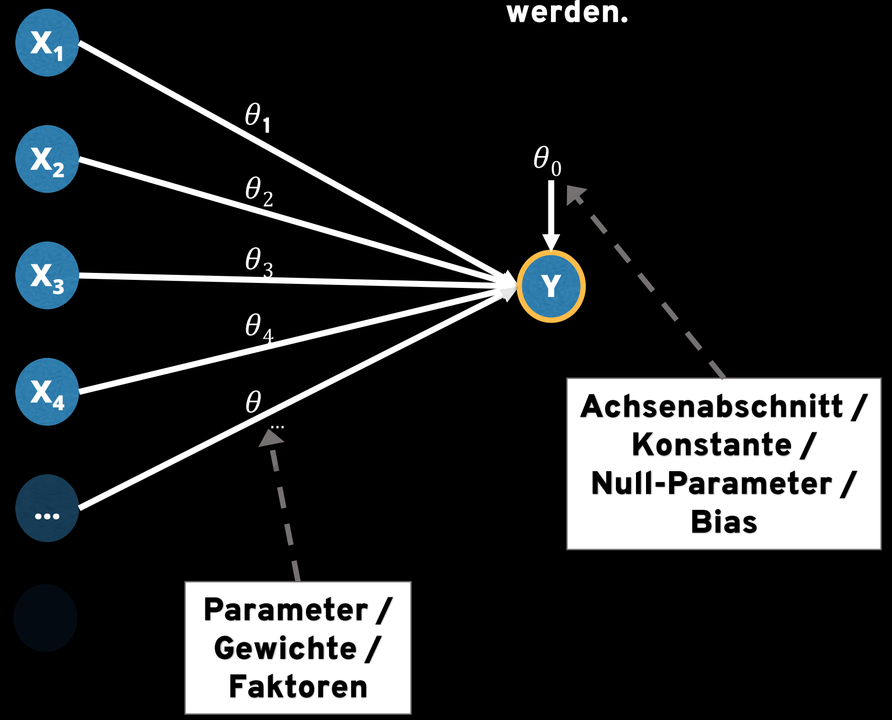
**Kostenfunktion mit Regularisierung:**

$$J_x(\theta_0, \theta_1, \theta_2, \dots) = \frac{1}{m}\left[\sum_m (h_x(\theta_0, \theta_1, \theta_2, \dots) - y)^2 \right]$$

# ZUSAMMENFASSUNG LINEARE REGRESSION

**Input Layer**
Elemente sind die Input-Variablen; auch genannt: Input-Features oder Input-Dimensionen.

**Output Layer**
Nutzt eine *„Aktivierungsfunktion“* (hier lineare Funktion) mit der die Parameter $\theta$ der eingehenden Schicht zusammengefasst werden.




- Ziel ist, anhand des Trainingsdatensatzes die Parameter $\theta$ der Aktivierungsfunktion für eine bestmögliche Vorhersage des Testdatensatzes zu optimieren.

- Die Optimierung mit Regularisierung erlaubt, viele Variablen in das Modell eingehen zu lassen und über einen Regularisierungs- (oder Shrinkage-) Parameter den Umfang des Einsatzes der Variablen zu kontrollieren, um so Over-/ Underfitting zu kontrollieren.

# WICHTIGE HYPERPARAMETER

- Wahl des Modells bzw. der Modellarchitektur
- Art der Aktivierungsfunktionen („Vorhersagefunktionen“)
- Art der Kostenfunktion
  - mit oder ohne Regularisierung
  - Höhe des Regularisierungsparameters
- Art der Optimierungsfunktion (zur Minimierung der Kostenfunktion)
  - Höhe der Lernrate
- Je nach Modellarchitektur zahlreiche weitere…

# EIGENSCHAFTEN DES LINEAREN MODELLS

- Die Funktion lm() liefert optimierte Parameter für das lineare Modell ohne Regularisierung (Angabe eines Lernparameters ist hier nicht nötig)

  → Für einfache Modelle ist es einfach optimierte Parameter zu erhalten.

- Einfacher zu schätzende Modelle haben stärkere Annahmen über den Zusammenhang der Variablen (hier linearer Zusammenhang)

  → Die optimale Kodierung/Kategorisierung der Variablen entsprechend der Annahmen ist umso wichtiger.

# ABRUFEN DES MAPE FÜR DEN TESTDATENSATZ

```r
library(dplyr)
library(readr)
library(httr)

# Dataframe for request must include columns `Datum`, `Warengruppe` und `Umsatz`
predictions <- read_csv("prediction_template.csv")

# name must not be provided; however, each team must upload at least one not
# anonymous prediction
name <- "Gruppe X"

# Execution of the request
r <- POST("https://bakery-sales-mape-tolicqztoq-ey.a.run.app/",
          body = list(name=name, predictions=predictions),
          encode = "json")
# Output of MAPE in Percent
content(r, "parsed", "application/json")
```

# AUFGABEN

- Datensatz weiter um zusätzliche Variablen ergänzen, die für die Schätzung des Umsatzes relevant sein könnten.
- Modellgleichung aufstellen, die das adjustierte $R^2$ auf Basis eines linearen Modells für Euren Datensatz maximiert.
- Zum Thema Overfitting dieses Video (9 Minuten) anschauen.
- Euch dieses Video (12 Minuten) zur Einführung in Neuronale Netze an anschauen.
- Einmal dieses R-Script durchlaufen lassen, um Python (bzw. Miniconda) mit verschiedenen zusätzlichen Funktionspaketen zu installieren.