Datamanagement & Analytics

Prof. Dr. André Drews

Fachbereich Maschinenbau und Wirtschaft

22. Juni 2026

- nur für den Gebrauch innerhalb des Kurses -

TECHNISCHE
HOCHSCHULE
LÜBECK

Vorlesungskapitel 10

- Absatzprognose und Dashboarding -

Lernziele

- Kompetenz in Verfahren der Absatzprognoseverfahren
- Kompetenz in Dashboarding

Table Of Contents

1. Absatzprognose
2. Dashboarding in PowerBI und Superset

Absatzprognose, Praxisbeispiele

Absatzprognose - Qualitative Verfahren bei fehlenden quantitativen Daten

Verfahren:

- Delphi-Methode
- Vertriebsmitarbeiterschätzung
- Kundenbefragung
- Szenarioanalyse

Vorteil:

- Berücksichtigt Expertenwissen/Marktwissen abseits erhobener quantitativer Daten

Nachteile:

- subjektiv, schwer reproduzierbar

Typischer Ablauf der Delphi-Methode

1. Expertenauswahl
   - Zusammenstellung eines diversen Expertengremiums
   - Beispielsweise:
     - Vertriebsleiter
     - Marktforscher
     - Branchenanalysten
     - wichtige Kunden
2. Erste Befragung
   - Schriftliche und anonyme Abgabe einer ersten Absatzschätzung
3. Auswertung und Feedback
   - Sammlung und Auswertung der Antworten durch einen Moderator
   - Ermittlung von Mittelwerten und Streuungen
   - Aufbereitung der Bandbreite der Schätzungen

Typischer Ablauf der Delphi-Methode

4. Folgebefragungen (Iterationen)
   - Experten erhalten die anonymisierten Ergebnisse der vorherigen Runde
   - Starke Abweichungen vom Mittelwert müssen begründet oder angepasst werden
5. Konsensfindung
   - Wiederholung des Prozesses über mehrere Runden
   - Ziel: Stabilisierung der Prognosen bzw. Erreichen eines Konsenses über den erwarteten Absatz

Vorteil

Durch die anonyme Befragung werden Gruppendruck und Mitläufereffekte weitgehend vermieden.

Szenarioanalyse

- Methode zur Abschätzung zukünftiger Entwicklungen unter verschiedenen Annahmen
- Berücksichtigt Unsicherheiten und mögliche Veränderungen im Umfeld
- Entwicklung mehrerer realistischer Zukunftsszenarien
- Typischerweise werden drei Szenarien betrachtet:
  - Best-Case-Szenario (optimistische Entwicklung)
  - Realistic-Case-Szenario (wahrscheinlichste Entwicklung)
  - Worst-Case-Szenario (pessimistische Entwicklung)
- Analyse der Auswirkungen auf Absatz, Umsatz, Produktion oder Logistik
- Unterstützung bei strategischen Entscheidungen und Risikobewertungen

Absatzprognose - Klassische, quantitative Verfahren / Zeitreihenverfahren

- Einfache Zeitreihenglättungen (Gleitender Mittelwert, Exponentielle Glättung)
- Zeitreihenanalyse: Zeitreihenkompontenverfahren, Autoregression
- ARIMA Verfahren

Einfache Zeitreihenglättungen

- Ziel:
  - Glättung zufälliger Schwankungen in Zeitreihen
  - Verbesserung der Prognosequalität
- Einsatzgebiete:
  - Absatzprognosen
  - Lagerbestandsplanung
  - Nachfrageprognosen
  - Transport- und Kapazitätsplanung
- Grundidee:
  - Vergangene Beobachtungen werden genutzt, um zukünftige Werte vorherzusagen.
  - Zufällige Schwankungen werden reduziert.

Gleitender Mittelwert

- Prognose basiert auf dem Durchschnitt der letzten n Beobachtungen.
- Alle berücksichtigten Werte werden gleich gewichtet.
- Geeignet für Zeitreihen ohne ausgeprägten Trend oder Saisonalität.

Prognosegleichung:

$$\hat{y}_{t+1} = \frac{1}{n} \sum_{i=0}^{n-1} y_{t-i}$$

mit

- y_t: Beobachtungswert zum Zeitpunkt t
- n: Anzahl berücksichtigter Perioden
- $\hat{y}_{t+1}$: Prognosewert für die nächste Periode

Beispiel: Gleitender Mittelwert

Gegeben seien die letzten drei Beobachtungen:

$$y_{t-2} = 100, \quad y_{t-1} = 120, \quad y_t = 110$$

Bei einem gleitenden Mittelwert mit $n = 3$ ergibt sich:

$$\hat{y}_{t+1} = \frac{100 + 120 + 110}{3} = 110$$

Eigenschaften:

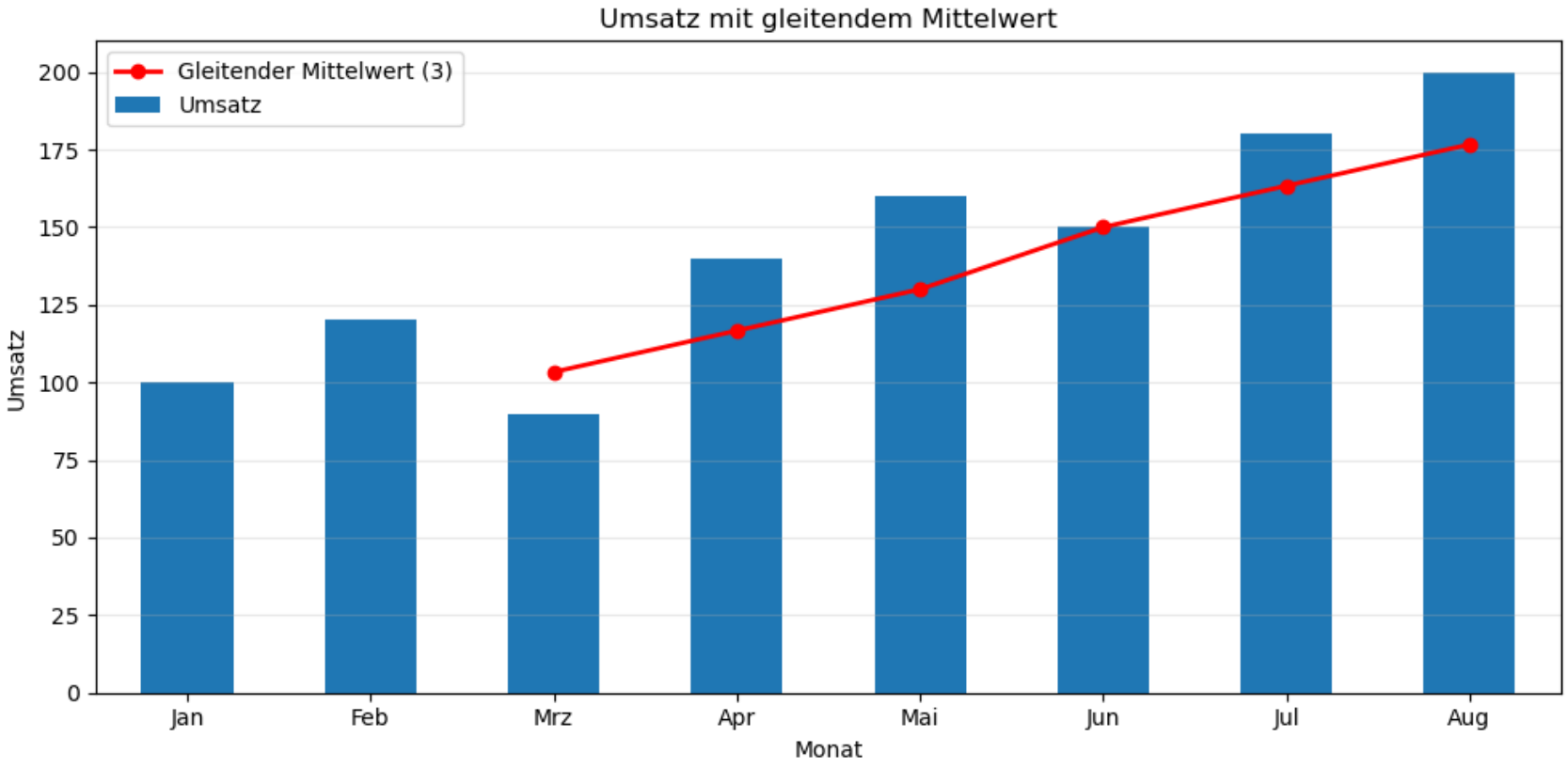
- Einfach zu berechnen
- Gute Glättung zufälliger Schwankungen
- Reagiert verzögert auf Strukturänderungen

Gleitender Mittelwert

```python
import pandas as pd
import matplotlib.pyplot as plt

# Beispieldaten
df = pd.DataFrame({
    'Monat': ['Jan', 'Feb', 'Mrz', 'Apr', 'Mai', 'Jun', 'Jul', 'Aug'],
    'Umsatz': [100, 120, 90, 140, 160, 150, 180, 200]
})

# Gleitender Mittelwert über 3 Perioden
df['MA_3'] = df['Umsatz'].rolling(window=3).mean()

# Diagramm erstellen
ax = df.plot(
    x='Monat',
    y='Umsatz',
    kind='bar',
    figsize=(10, 5),
    legend=False
)

# Gleitenden Mittelwert als Linie hinzufügen
df.plot(
    x='Monat',
    y='MA_3',
    ax=ax,
    color='red',
    marker='o',
    linewidth=2,
    label='Gleitender Mittelwert (3)'
)

plt.title('Umsatz mit gleitendem Mittelwert')
plt.ylabel('Umsatz')
plt.grid(axis='y', alpha=0.3)
plt.legend()
plt.tight_layout()

plt.show()
```

- nur für den Gebrauch innerhalb der Vorlesung -

Gleitender Mittelwert



Exponentielle Glättung 1. Ordnung

- Ist-Wert: y_t
- Prognosewert: $\hat{y}_t$ mit $\hat{y}_0 = y_0$
- Neuere Beobachtungen erhalten ein höheres Gewicht.
- Ältere Werte verlieren exponentiell an Bedeutung.
- Geeignet für kurzfristige Prognosen ohne Trend und Saisonalität.

Prognosegleichung:

$$\hat{y}_{t+1} = \hat{y}_t + \alpha(y_t - \hat{y}_t)$$

mit

- $0 \leq \alpha \leq 1$
- α: Glättungsparameter

Beispiel: Exponentielle Glättung

Gegeben:

$$y_t = 120$$

$$\hat{y}_t = 110$$

$$\alpha = 0.3$$

Dann ergibt sich:

$$\hat{y}_{t+1} = 110 + 0.3 \cdot (120 - 110)$$

$$= 113$$

Die Prognose für die nächste Periode beträgt:

$$\boxed{\hat{y}_{t+1} = 113}$$

Einfluss des Glättungsparameters

$$\hat{y}_{t+1} = \alpha y_t + (1 - \alpha)\hat{y}_t$$

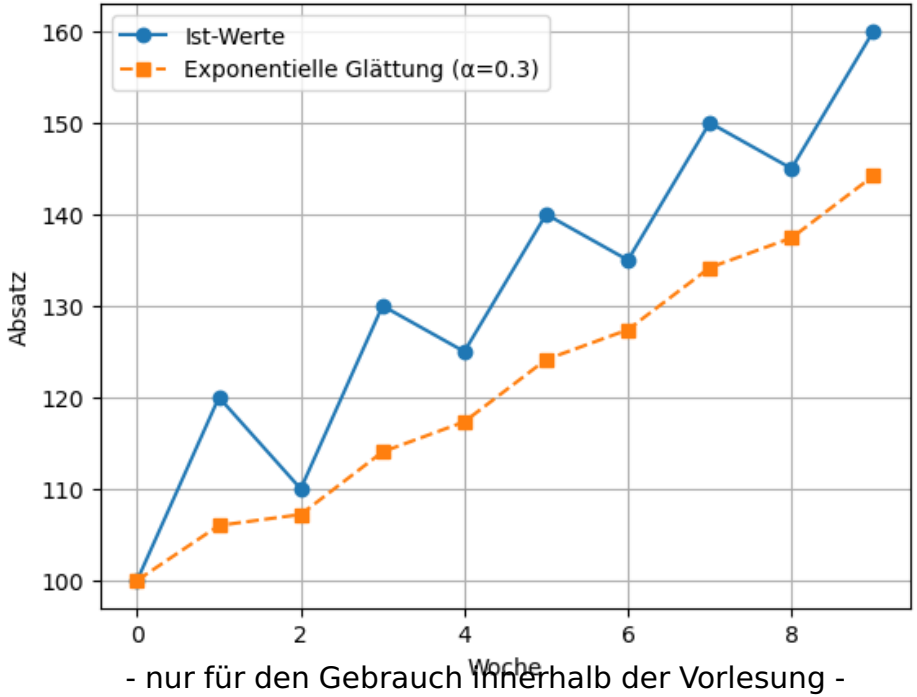
- Großes α (z. B. 0.8)
  - Hohe Gewichtung aktueller Beobachtungen
  - Schnelle Reaktion auf Veränderungen
- Kleines α (z. B. 0.1)
  - Stärkere Glättung
  - Stabilere Prognosen
  - Langsamere Reaktion auf Veränderungen

Vergleich der Verfahren

Gleitender Mittelwert	Exponentielle Glättung
Alle Werte gleich gewichtet	Neuere Werte stärker gewichtet
Feste Fenstergröße n	Glättungsparameter α
Einfache Berechnung	Flexiblere Anpassung
Träge bei Änderungen	Schnellere Reaktion auf Änderungen
Höherer Speicherbedarf	Geringer Speicherbedarf

Python Glättungsverfahren

```python
import pandas as pd
import matplotlib.pyplot as plt

# Absatzdaten
absatz = pd.Series([100, 120, 110, 130, 125, 140, 135, 150, 145, 160])

# Exponentielle Glättung
alpha = 0.3
glaettung = absatz.ewm(alpha=alpha, adjust=False).mean()

# Plot
plt.plot(absatz, "o-", label="Ist-Werte")
plt.plot(glaettung, "s--", label=f"Exponentielle Glättung (α={alpha})")

plt.xlabel("Woche")
plt.ylabel("Absatz")
plt.legend()
plt.grid()
plt.show()
```



Zeitreihenkomponentenverfahren

- Zerlegung einer Zeitreihe in ihre Bestandteile
- Ziel: Verständnis der Einflussfaktoren auf die Entwicklung der Zeitreihe

Additives Modell:

$$y_t = T_t + S_t + Z_t + R_t$$

Multiplikatives Modell:

$$y_t = T_t \cdot S_t \cdot Z_t \cdot R_t$$

mit

- T_t: Trendkomponente
- S_t: Saisonkomponente
- Z_t: Zykluskomponente
- R_t: Rest- bzw. Zufallskomponente

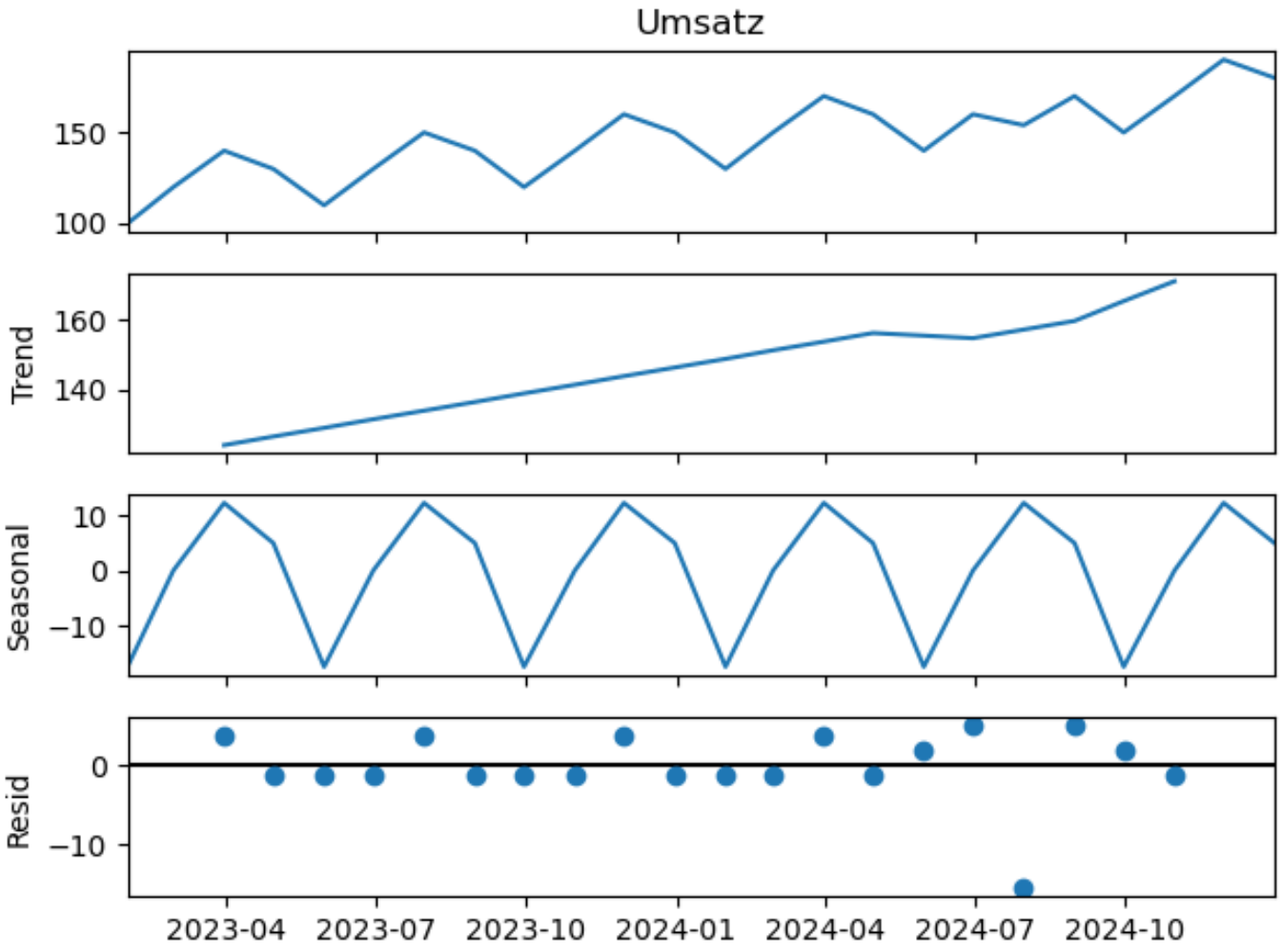
Komponenten einer Zeitreihe

- Trend
  - Langfristige Entwicklung
  - Beispiel: stetig steigender Online-Handel
- Saison
  - Regelmäßig wiederkehrende Schwankungen
  - Beispiel: Weihnachtsgeschäft
- Zyklus
  - Langfristige wirtschaftliche Schwankungen
  - Beispiel: Konjunkturzyklen
- Restkomponente
  - Zufällige Einflüsse
  - Nicht durch andere Komponenten erklärbar

Zeitreihen Dekomposition

```python
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose

# Beispiel-Zeitreihe
df = pd.DataFrame({
    'Umsatz': [
        100, 120, 140, 130,
        110, 130, 150, 140,
        120, 140, 160, 150,
        130, 150, 170, 160,
        140, 160, 154, 170,
        150, 170, 190, 180
    ]
})

# Datumsindex erzeugen
df.index = pd.date_range(
    start='2023-01-01',
    periods=len(df),
    freq='ME'
)

# Saisonale Zerlegung
result = seasonal_decompose(
    df['Umsatz'],
    model='additive',  # oder 'multiplicative'
    period=4           # Saisonlänge
)

# Standardplot
result.plot()
plt.tight_layout()
plt.show()
```

Zeitreihen Dekomposition



Autokorrelationsfunktion (ACF)

- Misst die lineare Abhängigkeit einer Zeitreihe zu ihren vergangenen Werten
- Hilft bei der Identifikation von Zeitreihenmodellen
- Wichtig für AR-, MA- und ARIMA-Modelle

Autokorrelation für Lag k:

$$\rho_k = \frac{Cov(y_t, y_{t-k})}{Var(y_t)}$$

Schätzung anhand von Stichprobendaten:

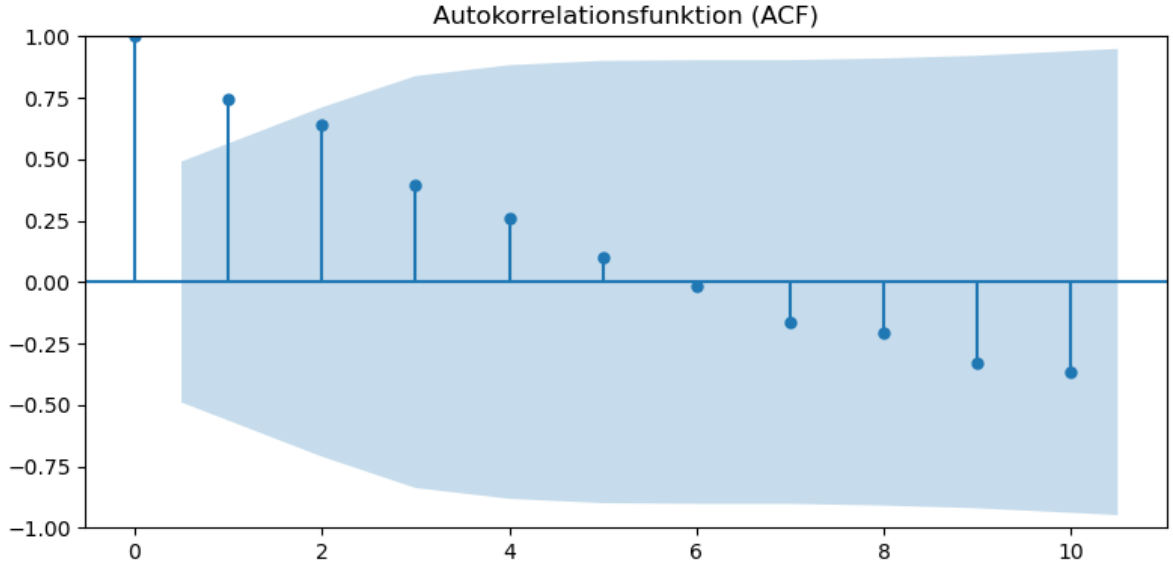
$$r_k = \frac{\sum_{t=k+1}^{n}(y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum_{t=1}^{n}(y_t - \bar{y})^2}$$

mit

- k: Lag (Zeitverzögerung)
- $\bar{y}$: Mittelwert der Zeitreihe
- r_k: geschätzte Autokorrelation

Autocorrelationfunktion

```python
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf

# Beispiel-Zeitreihe
df = pd.DataFrame({
    'Umsatz': [100, 120, 90, 140, 160, 150, 180, 200,
               190, 220, 210, 240, 230, 260, 250, 280]
})

# ACF-Plot
fig, ax = plt.subplots(figsize=(8, 4))

plot_acf(
    df['Umsatz'],
    lags=10,      # Anzahl der Lags
    ax=ax
)

plt.title('Autokorrelationsfunktion (ACF)')
plt.tight_layout()
plt.show()
```



Partielle Autokorrelationsfunktion (PACF)

- Misst die direkte Korrelation zwischen y_t und y_{t-k}
- Der Einfluss der dazwischenliegenden Lags wird herausgerechnet
- Wichtig zur Bestimmung der Ordnung p eines AR-Modells

Idee:

$$y_t \longleftrightarrow y_{t-k}$$

unter Kontrolle von

$$y_{t-1}, y_{t-2}, \ldots, y_{t-k+1}$$

Partielle Autokorrelation:

$$\phi_{kk} = Corr\left(y_t, y_{t-k} \mid y_{t-1}, \ldots, y_{t-k+1}\right)$$

mit

- k: Lag
- ϕ_{kk}: partielle Autokorrelation zum Lag k

Moving Average (MA)

- Modelliert eine Zeitreihe durch aktuelle und vergangene Fehlerterme
- Im Gegensatz zur Autoregression werden keine vergangenen Beobachtungen verwendet
- Geeignet zur Modellierung kurzfristiger zufälliger Schwankungen

MA(1)-Modell:

$$y_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1}$$

MA(q)-Modell:

$$y_t = \mu + \varepsilon_t + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}$$

mit

- μ: Mittelwert der Zeitreihe
- ε_t: aktueller Fehlerterm
- θ_i: MA-Koeffizienten
- q: Ordnung des MA-Modells

Autoregression (AR)

- Zukünftige Werte werden aus vergangenen Beobachtungen geschätzt.
- Annahme:
  - Vergangene Werte enthalten Informationen über zukünftige Werte.

AR(1)-Modell:

$$y_t = c + \phi_1 y_{t-1} + \varepsilon_t$$

AR(p)-Modell:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t$$

mit

- c: Konstante
- ϕ_i: Modellparameter
- ε_t: Zufallsfehler

Beispiel einer Autoregression

Gegeben:

$$c = 10$$

$$\phi_1 = 0.8$$

$$y_{t-1} = 100$$

Dann ergibt sich:

$$y_t = 10 + 0.8 \cdot 100 = 90$$

Interpretation:

- Der aktuelle Wert hängt stark vom vorherigen Wert ab.
- Typisch für Absatz-, Nachfrage- und Lagerbestandsdaten.

Stationarität

- Wichtige Voraussetzung vieler Zeitreihenmodelle
- Eine stationäre Zeitreihe besitzt konstante statistische Eigenschaften.

Schwache Stationarität:

$$E(y_t) = \mu$$

$$Var(y_t) = \sigma^2$$

$$Cov(y_t, y_{t-k}) = \gamma(k)$$

Eigenschaften:

- Konstanter Mittelwert
- Konstante Varianz
- Kovarianz nur abhängig vom Lag k

Nicht-stationäre Zeitreihen

Typische Ursachen:

- Trend
- Saisonale Schwankungen
- Strukturbrüche
- Veränderliche Varianz

Beispiel:

$$100,\ 105,\ 110,\ 115,\ 120, \ldots$$

- Mittelwert steigt kontinuierlich
- Zeitreihe ist nicht stationär

Lösung:

- Trendbereinigung
- Saisonbereinigung
- Differenzbildung

Differenzbildung zur Stationarisierung

Erste Differenz:

$$\nabla y_t = y_t - y_{t-1}$$

Beispiel:

$$100,\ 105,\ 110,\ 115,\ 120$$

wird zu

$$5,\ 5,\ 5,\ 5$$

Vorteile:

- Entfernung von Trends
- Herstellung von Stationarität
- Grundlage für ARIMA-Modelle

ARIMA-Verfahren

- ARIMA = AutoRegressive Integrated Moving Average
- Eines der bekanntesten Modelle der klassischen Zeitreihenanalyse
- Kombination aus:
  - Autoregression (AR)
  - Differenzbildung (I = Integrated)
  - Moving Average (MA)
- Geeignet für stationäre oder stationarisierte Zeitreihen

Bestandteile des ARIMA-Modells

Ein ARIMA-Modell wird durch drei Parameter beschrieben:

$$ARIMA(p, d, q)$$

mit

- p: Ordnung der Autoregression (AR)
- d: Anzahl der Differenzbildungen
- q: Ordnung des Moving-Average-Teils (MA)

Beispiel:

$$ARIMA(2, 1, 1)$$

- 2 autoregressive Terme
- 1 Differenzbildung
- 1 Moving-Average-Term

Autoregressiver Anteil (AR)

Die aktuellen Werte hängen von vergangenen Beobachtungen ab.

AR(1)-Modell:

$$y_t = c + \phi_1 y_{t-1} + \varepsilon_t$$

AR(p)-Modell:

$$y_t = c + \sum_{i=1}^{p} \phi_i y_{t-i} + \varepsilon_t$$

mit

- ϕ_i: autoregressive Koeffizienten
- ε_t: Zufallsfehler

Moving-Average-Anteil (MA)

Die aktuellen Werte hängen von vergangenen Fehlertermen ab.

MA(1)-Modell:

$$y_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1}$$

MA(q)-Modell:

$$y_t = \mu + \varepsilon_t + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j}$$

mit

- θ_j: MA-Koeffizienten
- ε_t: Fehlerterm

Integrated: Differenzbildung

Viele Zeitreihen sind nicht stationär.

Erste Differenz:

$$\nabla y_t = y_t - y_{t-1}$$

Beispiel:

$$100,\ 105,\ 110,\ 115,\ 120$$

wird zu

$$5,\ 5,\ 5,\ 5$$

Ziel:

- Entfernung von Trends
- Herstellung der Stationarität

Allgemeines ARIMA-Modell

Nach der Differenzbildung wird die stationäre Zeitreihe modelliert.

$$\Delta^d y_t = c + \sum_{i=1}^{p} \phi_i \Delta^d y_{t-i} + \varepsilon_t + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j}$$

Bestandteile:

- AR-Komponente: vergangene Werte
- I-Komponente: Differenzbildung
- MA-Komponente: vergangene Fehler

Beispiel: ARIMA(1,1,1)

Gegeben:

$$\Delta y_t = 0.7\Delta y_{t-1} + \varepsilon_t + 0.4\varepsilon_{t-1}$$

Interpretation:

- $p = 1$: ein autoregressiver Term
- $d = 1$: eine Differenzbildung
- $q = 1$: ein MA-Term

Die aktuelle Veränderung hängt ab von:

- der vorherigen Veränderung
- dem aktuellen Fehler
- dem vorherigen Fehler

Modellidentifikation

Zur Bestimmung von p und q werden verwendet:

- Autokorrelationsfunktion (ACF)
- Partielle Autokorrelationsfunktion (PACF)

Typische Vorgehensweise:

1. Stationarität prüfen
2. Differenzieren
3. ACF und PACF analysieren
4. ARIMA-Modell schätzen
5. Prognose erstellen

Vor- und Nachteile von ARIMA

Vorteile:

- Mathematisch gut fundiert
- Gute Ergebnisse bei linearen Zeitreihen
- Weit verbreitet und gut untersucht

Nachteile:

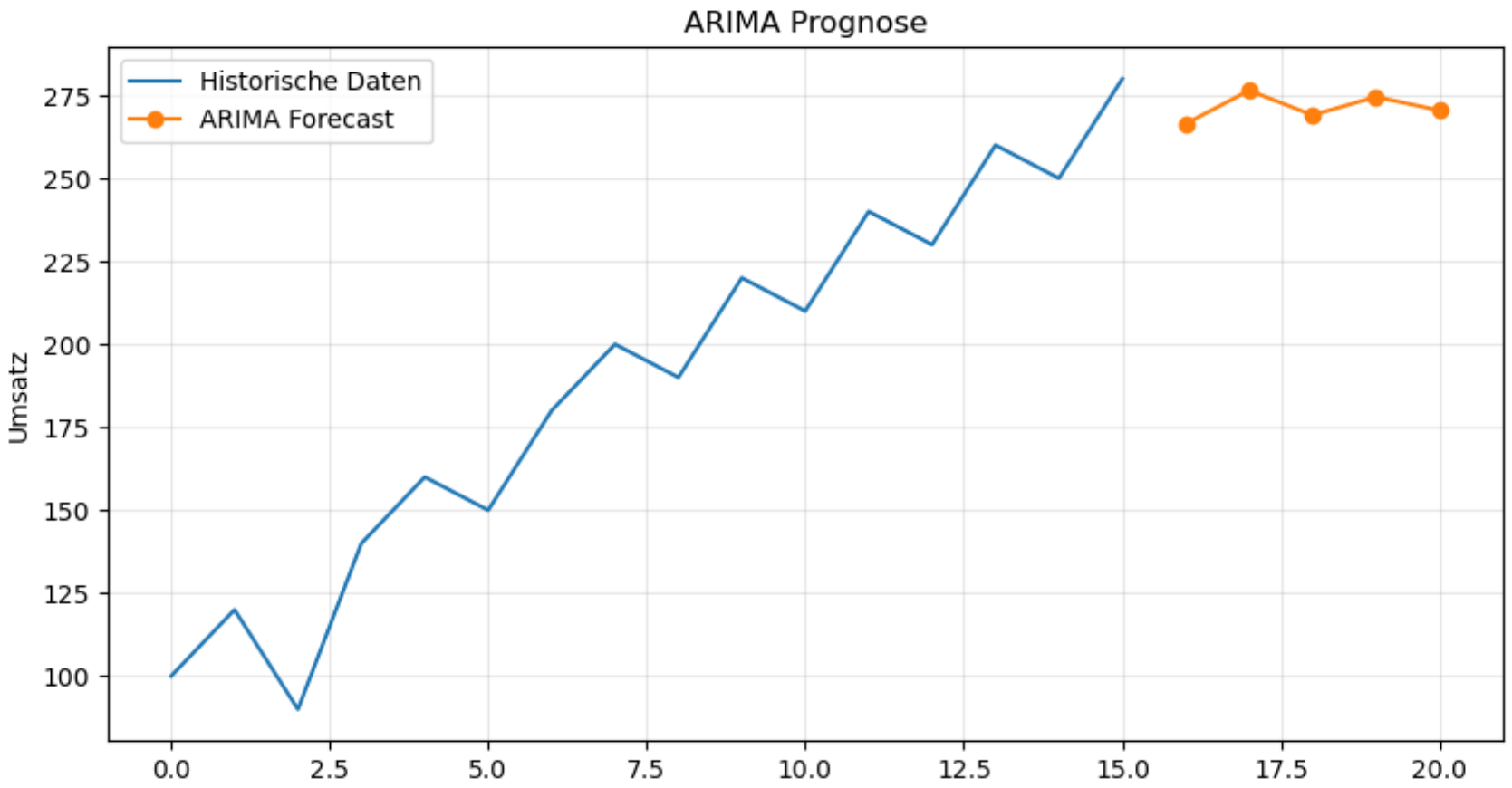
- Stationarität erforderlich
- Modellidentifikation teilweise aufwendig
- Schwierigkeiten bei nichtlinearen Zusammenhängen
- Begrenzte Leistungsfähigkeit bei sehr langen Abhängigkeiten

Arima

```python
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA

# Beispiel-Daten
df = pd.DataFrame({
    'Umsatz': [100, 120, 90, 140, 160, 150, 180, 200,
               190, 220, 210, 240, 230, 260, 250, 280]
})

# ARIMA(p,d,q)
model = ARIMA(df['Umsatz'], order=(1, 1, 1))

# Modell fitten
model_fit = model.fit()

# 5 Schritte in die Zukunft prognostizieren
forecast = model_fit.forecast(steps=5)


# Modell
model = ARIMA(df['Umsatz'], order=(1, 1, 1))
model_fit = model.fit()
```


```python
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA

# Beispiel-Daten
df = pd.DataFrame({
    'Umsatz': [100, 120, 90, 140, 160, 150, 180, 200,
               190, 220, 210, 240, 230, 260, 250, 280]
})

# ARIMA(p,d,q)
model = ARIMA(df['Umsatz'], order=(1, 1, 1))

# Modell fitten
model_fit = model.fit()

# 5 Schritte in die Zukunft prognostizieren
forecast = model_fit.forecast(steps=5)


# Modell
model = ARIMA(df['Umsatz'], order=(1, 1, 1))
model_fit = model.fit()
```

Arima


ARIMA Prognose
Historische Daten
ARIMA Forecast
Umsatz
275
250
225
200
175
150
125
100
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

Absatzprognose - Klassisches Maschinelles Lernen - Regressionsverfahren

Voraussetzungen:

- Historische Daten (gelabelte Daten)
- $Y(X)$ mit abhängiger Variable Y und unabhängiger Variablen X

Regressionsmodelle:

- Multilineare Regression
- Ridge-Regression
- Lass-Regression

Regressionsverfahren zur Absatzprognose

- Ziel:
  - Prognose des zukünftigen Absatzes anhand mehrerer Einflussgrößen
- Mögliche Einflussgrößen:
  - Preis
  - Werbeausgaben
  - Saison
  - Anzahl der Verkaufsstellen
  - Wettbewerbsaktivitäten
- Verfahren:
  - Multilineare Regression
  - Ridge-Regression
  - Lasso-Regression

Beispiel einer Absatzprognose

Zu prognostizieren ist der Absatz y.

$$f(x_1, x_2, x_3, x_4) \rightarrow \hat{y}$$

Einflussgrößen:

$$x_1 = \text{Preis}, x_2 = \text{Werbebudget}, x_3 = \text{Anzahl Verkaufsstellen}$$

$$, x_4 = \text{Saisonindex}$$

Ziel:

- Schätzung des zukünftigen Absatzes $\hat{y}$
- Berücksichtigung mehrerer Einflussfaktoren x_i

Multilineare Regression

Die multilineare Regression modelliert den Zusammenhang zwischen dem Absatz und mehreren Einflussgrößen.

Modell:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

Beispiel:

$$\hat{y} = 500 - 20 \cdot \mathit{Preis} + 0.8 \cdot \mathit{Werbung} + 15 \cdot \mathit{Verkaufsstellen}$$

Interpretation:

- Höherer Preis reduziert den Absatz
- Mehr Werbung erhöht den Absatz
- Mehr Verkaufsstellen steigern den Absatz

Multilineare Regression

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error

# ---------------------------------------------------
# Beispieldaten erzeugen
# ---------------------------------------------------

np.random.seed(42)

n = 100

df = pd.DataFrame({
    "Werbung": np.random.randint(10, 100, n),
    "Preis": np.random.randint(50, 120, n)
})

# Zielvariable mit etwas Zufallsrauschen
df["Umsatz"] = (
    5 * df["Werbung"]
    - 3 * df["Preis"]
    + 500
    + np.random.normal(0, 30, n)
)

# ---------------------------------------------------
# Features und Zielvariable
# ---------------------------------------------------

X = df[["Werbung", "Preis"]]
y = df["Umsatz"]
```

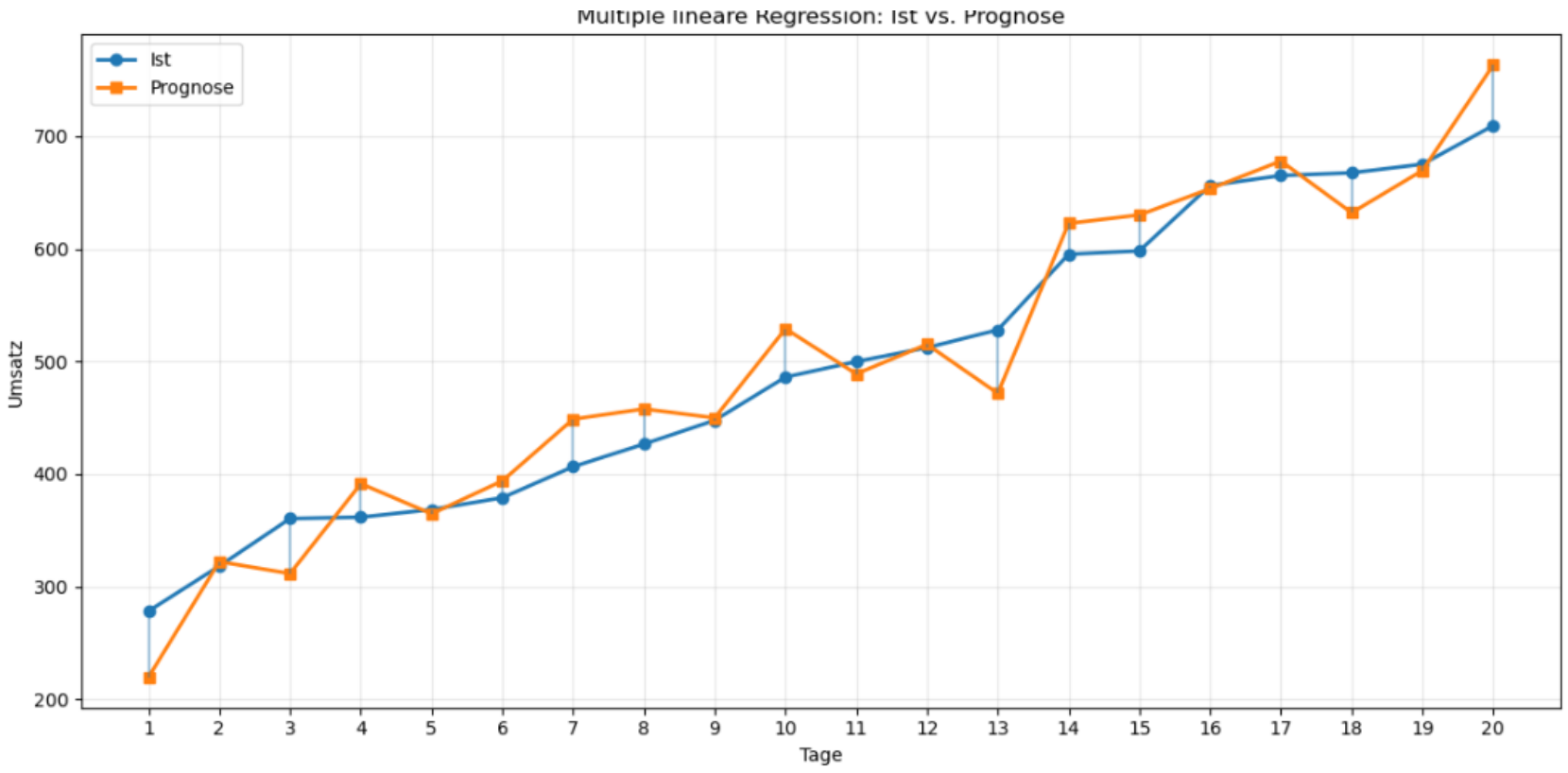

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error

# ---------------------------------------------------
# Beispieldaten erzeugen
# ---------------------------------------------------

np.random.seed(42)

n = 100

df = pd.DataFrame({
    "Werbung": np.random.randint(10, 100, n),
    "Preis": np.random.randint(50, 120, n)
})

# Zielvariable mit etwas Zufallsrauschen
df["Umsatz"] = (
    5 * df["Werbung"]
    - 3 * df["Preis"]
    + 500
    + np.random.normal(0, 30, n)
)

# ---------------------------------------------------
# Features und Zielvariable
# ---------------------------------------------------

X = df[["Werbung", "Preis"]]
y = df["Umsatz"]
```

Multilineare Regression

```python
# ---------------------------------------------------
# Plot
# ---------------------------------------------------

fig, ax = plt.subplots(figsize=(12, 6))

# Ist-Werte
ax.plot(
    result.index,
    result["Ist"],
    marker="o",
    linewidth=2,
    label="Ist"
)

# Prognose-Werte
ax.plot(
    result.index,
    result["Prognose"],
    marker="s",
    linewidth=2,
    label="Prognose"
)

# Fehlerbalken zwischen Ist und Prognose
for i in range(len(result)):
    ax.vlines(
        x=i,
        ymin=result["Ist"].iloc[i],
        ymax=result["Prognose"].iloc[i],
        alpha=0.4
    )

ax.set_title("Multiple lineare Regression: Ist vs. Prognose")
ax.set_xlabel("Tage")
ax.set_ylabel("Umsatz")

ax.set_xticks(result.index)
ax.set_xticklabels(result.index + 1)

ax.grid(alpha=0.3)
ax.legend()

plt.tight_layout()
plt.show()
```

```python
# Fehlerbalken zwischen Ist und Prognose
for i in range(len(result)):
    ax.vlines(
        x=i,
        ymin=result["Ist"].iloc[i],
        ymax=result["Prognose"].iloc[i],
        alpha=0.4
    )

ax.set_title("Multiple lineare Regression: Ist vs. Prognose")
ax.set_xlabel("Tage")
ax.set_ylabel("Umsatz")

ax.set_xticks(result.index)
ax.set_xticklabels(result.index + 1)

ax.grid(alpha=0.3)
ax.legend()

plt.tight_layout()
plt.show()

# ---------------------------------------------------
# Kennzahlen
# ---------------------------------------------------

print("\nModellgüte")
print("-" * 30)
print(f"R²  : {r2_score(y_test, pred):.3f}")
print(f"MAE : {mean_absolute_error(y_test, pred):.2f}")

# ---------------------------------------------------
# Beispiel für neue Daten
# ---------------------------------------------------

neu = pd.DataFrame({
    "Werbung": [60, 75, 90],
    "Preis": [80, 75, 70]
})

neu["Prognose_Umsatz"] = model.predict(neu)

print("\nNeue Vorhersagen")
print("-" * 30)
```

Multilineare Regression




```
Modellgüte
-----------------------------
R²  : 0.939
MAE : 25.93

Neue Vorhersagen
-----------------------------
   Werbung  Preis  Prognose_Umsatz
0       60     80       559.184654
1       75     75       649.306562
2       90     70       739.428471
```

Probleme der klassischen Regression

Mögliche Herausforderungen:

- Viele Einflussgrößen
- Starke Korrelationen zwischen Variablen
- Überanpassung (Overfitting)

Beispiel:

- Werbeausgaben TV
- Werbeausgaben Online
- Werbeausgaben Print

Diese Variablen sind häufig stark miteinander korreliert.

Lösung:

- Ridge-Regression
- Lasso-Regression

Ridge-Regression

Idee:

- Bestrafung großer Regressionskoeffizienten
- Verringerung von Overfitting
- Stabilere Modelle bei korrelierten Variablen

Zielfunktion:

$$\min_{\beta} \left(\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right)$$

mit

- λ: Regularisierungsparameter
- große Koeffizienten werden bestraft

Ridge-Regression bei der Absatzprognose

Beispiel:
Preis, TV-Werbung, Online-Werbung, Print-Werbung

Ohne Ridge:

$$\beta = (120, -90, 150, -110)$$

Mit Ridge:

$$\beta = (80, -45, 90, -60)$$

Vorteil:

- Stabilere Prognosen
- Geringere Varianz
- Bessere Generalisierung

Lasso-Regression

Idee:

- Bestrafung großer Koeffizienten
- Automatische Variablenselektion

Zielfunktion:

$$\min_{\beta} \left(\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right)$$

Unterschied zur Ridge-Regression:

- Ridge reduziert Koeffizienten
- Lasso kann Koeffizienten exakt auf Null setzen

Lasso-Regression bei der Absatzprognose

Ausgangsvariablen:
Preis, TV-Werbung, Online-Werbung, Print-Werbung, Wetter,, Wettbewerberpreis

Ergebnis:

$$\hat{y} = 450 - 15 \cdot \mathit{Preis} + 0.7 \cdot \mathit{OnlineWerbung} + 12 \cdot \mathit{Verkaufsstellen}$$

Lasso setzt beispielsweise:

$$\beta_{\text{TV}} = 0$$

$$\beta_{\text{Print}} = 0$$

Vorteil:

- Automatische Auswahl relevanter Einflussgrößen
- Einfachere Interpretation

Vergleich der Verfahren

Eigenschaft	Ridge	Lasso
Regularisierung	L_2	L_1
Reduziert Koeffizienten	Ja	Ja
Setzt Koeffizienten auf Null	Nein	Ja
Variablenselektion	Nein	Ja
Umgang mit Multikollinearität	Gut	Gut

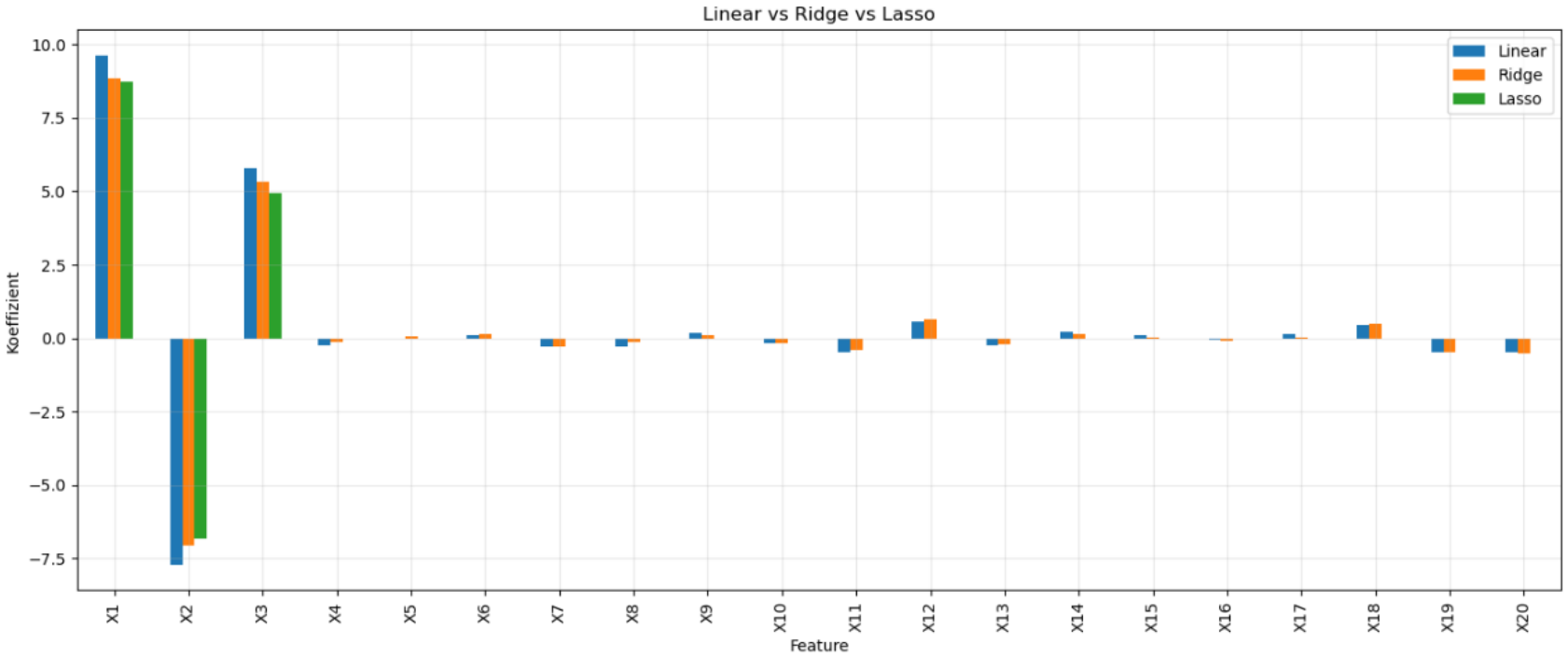
Multilineare Regression bildet die Grundlage, Ridge und Lasso erweitern sie um Regularisierung und verbessern die Prognosequalität bei vielen Einflussgrößen.

Lasso/Ridge

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso

# ---------------------------------------------------
# Datensatz erzeugen
# ---------------------------------------------------

np.random.seed(42)

n = 300

# 20 Features
X = pd.DataFrame({
    f"X{i}": np.random.normal(0, 1, n)
    for i in range(1, 21)
})

# Nur 3 Features sind wirklich relevant
y = (
    10 * X["X1"]
    - 8 * X["X2"]
    + 6 * X["X3"]
    + np.random.normal(0, 5, n)
)

# ---------------------------------------------------
# Train / Test
# ---------------------------------------------------

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.3,
    random_state=42
)
```

```python
# ---------------------------------------------------
# Modelle
# ---------------------------------------------------

linear = LinearRegression()
ridge = Ridge(alpha=20)
lasso = Lasso(alpha=1)

linear.fit(X_train, y_train)
ridge.fit(X_train, y_train)
lasso.fit(X_train, y_train)

# ---------------------------------------------------
# Koeffizientenvergleich
# ---------------------------------------------------

coef_df = pd.DataFrame({
    "Feature": X.columns,
    "Linear": linear.coef_,
    "Ridge": ridge.coef_,
    "Lasso": lasso.coef_
})

print(coef_df.round(2))

# ---------------------------------------------------
# Plot
# ---------------------------------------------------

coef_df.set_index("Feature").plot(
    kind="bar",
    figsize=(14, 6)
)

plt.title("Linear vs Ridge vs Lasso")
plt.ylabel("Koeffizient")
plt.xlabel("Feature")
plt.grid(alpha=0.3)

plt.tight_layout()
plt.show()
```

Lasso/Ridge


Linear vs Ridge vs Lasso
Linear
Ridge
Lasso
Koeffizient
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X11
X12
X13
X14
X15
X16
X17
X18
X19
X20
Feature

Absatzprognose - Klassisches Maschinelles Lernen - Regressionsverfahren

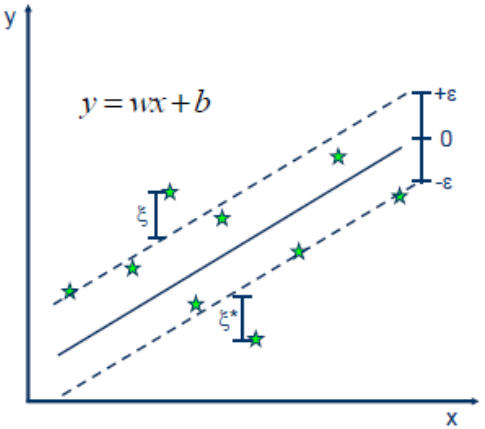
Voraussetzungen:

- Historische Daten (gelabelte Daten)
- $Y(X)$ mit abhängiger Variable Y und unabhängiger Variablen X

Methoden:

- SVR
- Entscheidungsbaum basierte Verfahren
- Gradient Boosting

Support Vector Regression (SVR)



- Minimize:

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*)$$

- Constraints:

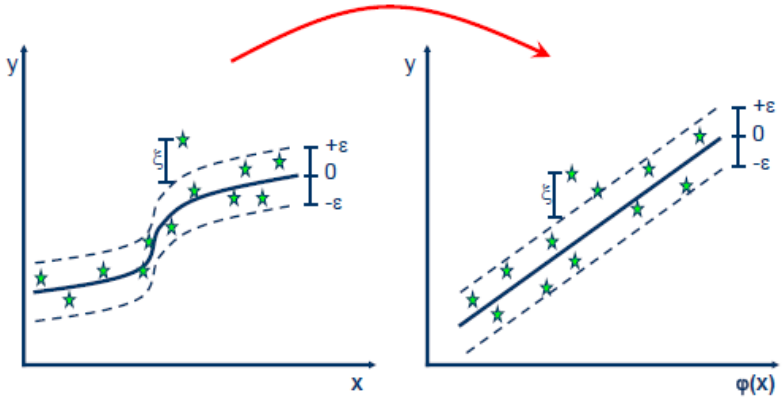
$$y_i - wx_i - b \le \varepsilon + \xi_i$$

$$wx_i + b - y_i \le \varepsilon + \xi_i^*$$

$$\xi_i, \xi_i^* \ge 0$$

Linear SVR

$$y = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\cdot\langle x_i, x\rangle + b$$

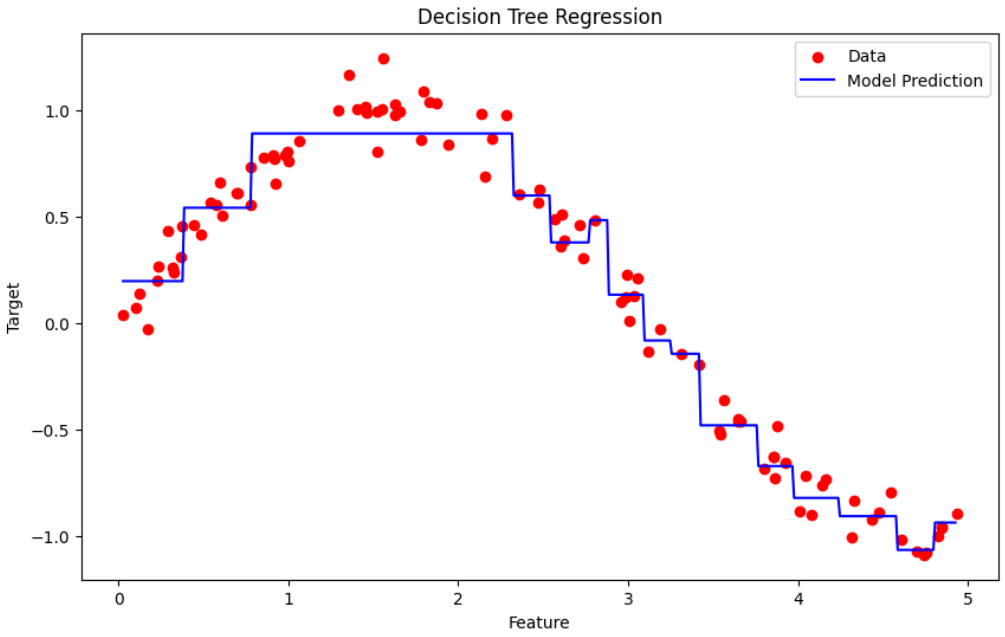


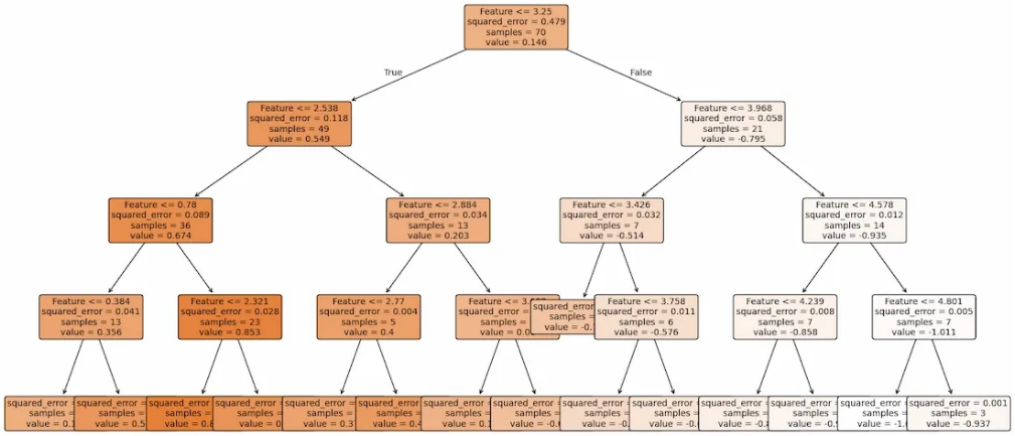
Kernel functions

Polynomial

$$k(\mathrm{x}_i, \mathrm{x}_j) = (\mathrm{x}_i.\mathrm{x}_j)^d$$

Entscheidungsbaums





Gradient Boosting

- Ensemble-Verfahren aus vielen Entscheidungsbäumen
- Bäume werden nacheinander trainiert
- Jeder neue Baum korrigiert die Fehler des vorherigen Modells

Modell:

$$F_M(x) = \sum_{m=1}^{M} \gamma_m h_m(x)$$

mit

- $h_m(x)$: Entscheidungsbaum
- γ_m: Gewichtung
- M: Anzahl der Bäume

Gradient Boosting bei der Absatzprognose

Eingangsgrößen:

- Preis
- Werbung
- Saison
- Wetter
- Wettbewerberpreise

Trainingsablauf:

1. Erster Baum erstellt grobe Absatzprognose.
2. Zweiter Baum modelliert die verbleibenden Fehler.
3. Weitere Bäume reduzieren die Restfehler schrittweise.

Endgültige Prognose:

$$\hat{y} = \sum_{m=1}^{M} \gamma_m h_m(x)$$

Bekannte Gradient-Boosting-Verfahren

- Gradient Boosting Machine (GBM)
- XGBoost
- LightGBM
- CatBoost

Eigenschaften:

- Hohe Prognosegenauigkeit
- Verarbeitung komplexer nichtlinearer Zusammenhänge
- Automatische Berücksichtigung von Merkmalsinteraktionen

Vergleich der Verfahren

Eigenschaft	SVR	Baum	Gradient Boosting
Nichtlinearität	Ja	Ja	Ja
Interpretierbarkeit	Mittel	Hoch	Niedrig
Prognosegüte	Hoch	Mittel	Sehr hoch
Overfitting-Risiko	Mittel	Hoch	Gering
Rechenaufwand	Mittel	Gering	Hoch

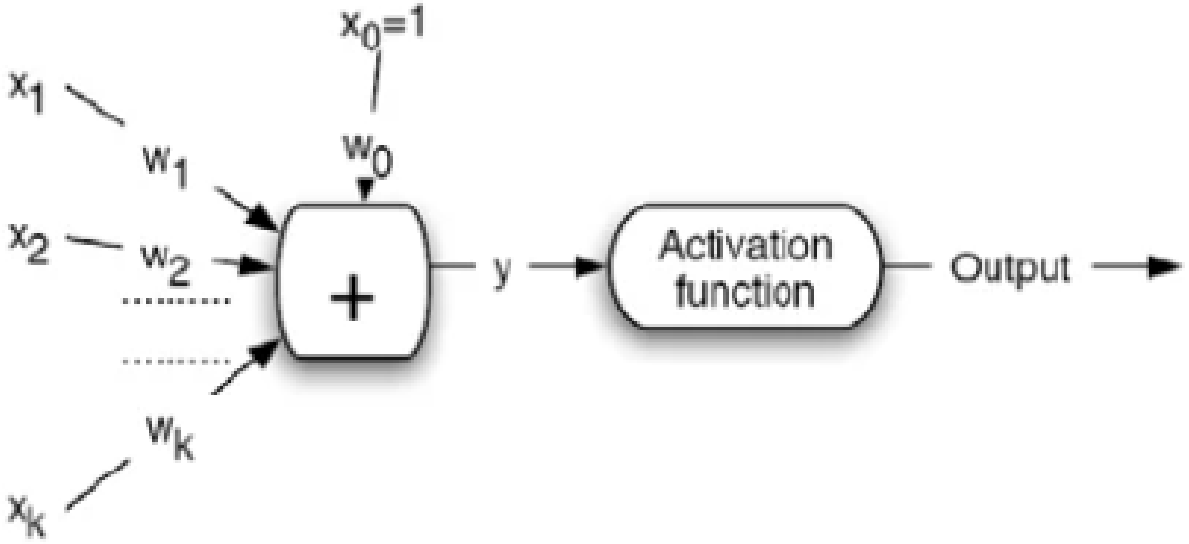
Gradient Boosting zählt heute zu den erfolgreichsten Verfahren für Absatz-, Nachfrage- und Logistikprognosen.

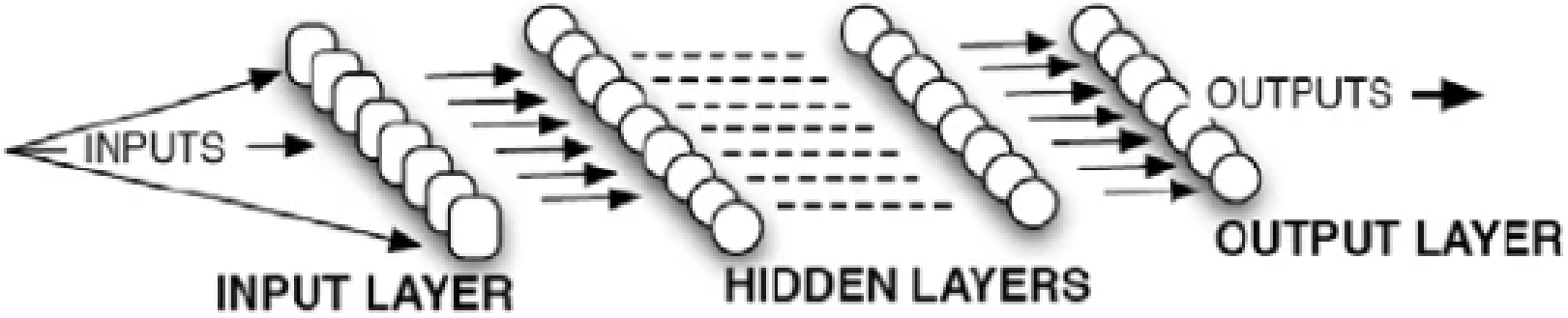
Absatzprognose - Deep Learning - ANN

Voraussetzungen: Große Datenmengen

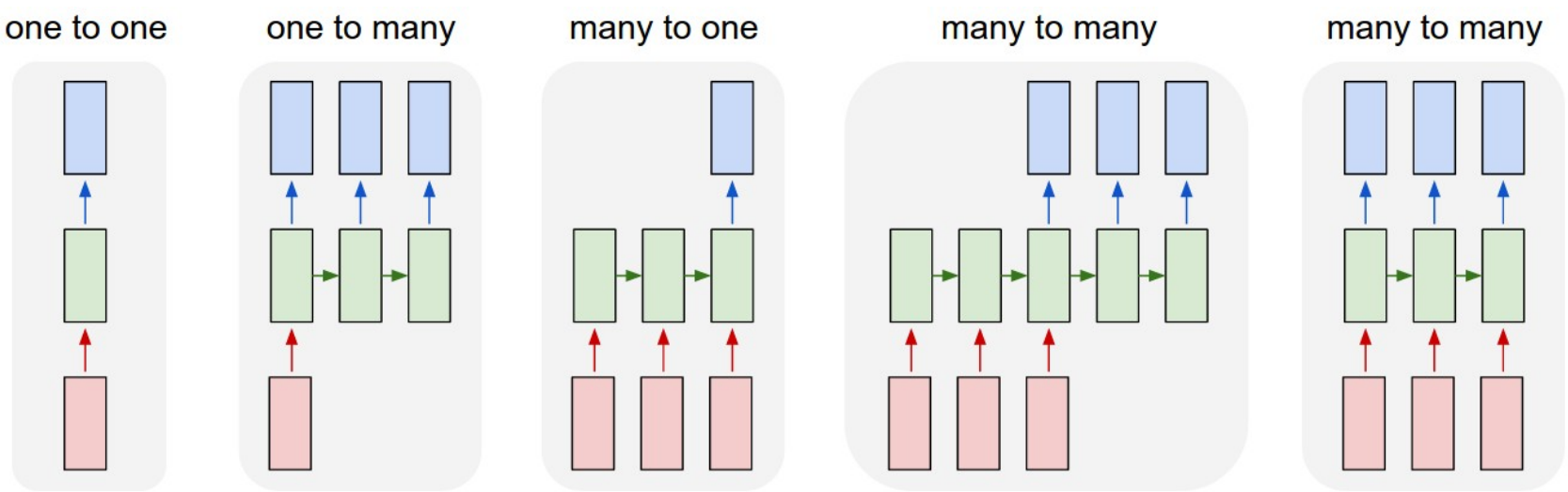
- RNN
- LSTM
- Transformer-Modelle

Grundlagen: Künstliche Neuronale Netze - MLP





Rekurrente Neuronale Netze



Recurrent Neural Networks (RNN)

- Speziell für sequentielle Daten entwickelt
- Frühere Informationen werden im versteckten Zustand gespeichert

Zustandsübergang:

$$h_t = f(W_h h_{t-1} + W_x x_t + b)$$

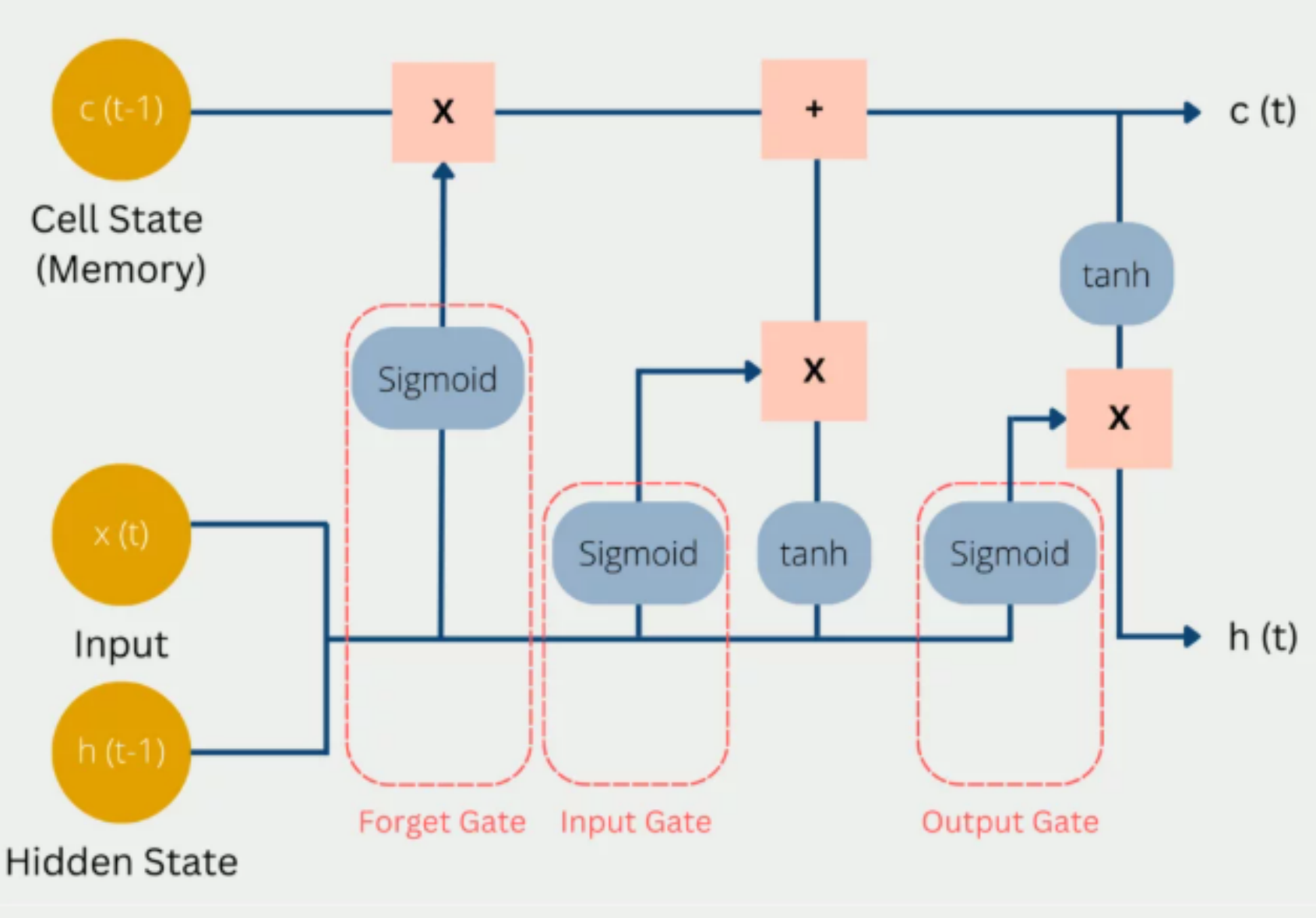
Ausgabe:

$$y_t = W_y h_t$$

mit

- x_t: Eingabe zum Zeitpunkt t
- h_t: versteckter Zustand
- y_t: Ausgabe

Long-Short-Term Memory LSTM



Long Short-Term Memory (LSTM)

- Erweiterung des RNN
- Speichert Informationen über lange Zeiträume
- Vermeidet das Vanishing-Gradient-Problem

LSTM-Zelle besitzt:

- Forget Gate
- Input Gate
- Output Gate
- Zellzustand c_t

Forget Gate:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

Zellzustand:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

LSTM bei der Absatzprognose

Beispiel:

- Historische Verkäufe der letzten 52 Wochen
- Saisonale Muster
- Werbeaktionen
- Feiertagseffekte

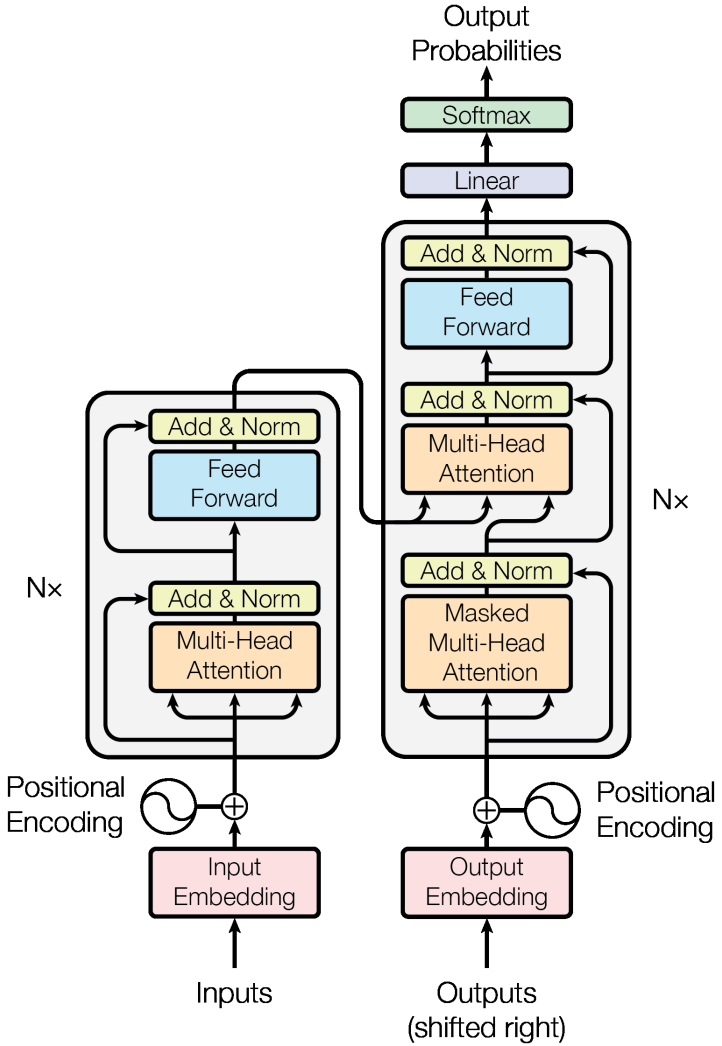
Input:
$(x_1, x_2, \ldots, x_{52})$
Output: $\hat{y}_{53}$

$$(\hat{y}_{53}, \hat{y}_{54}, \ldots, \hat{y}_{60})$$

Vorteile:

- Erkennung langfristiger Abhängigkeiten
- Berücksichtigung saisonaler Effekte

Transformer



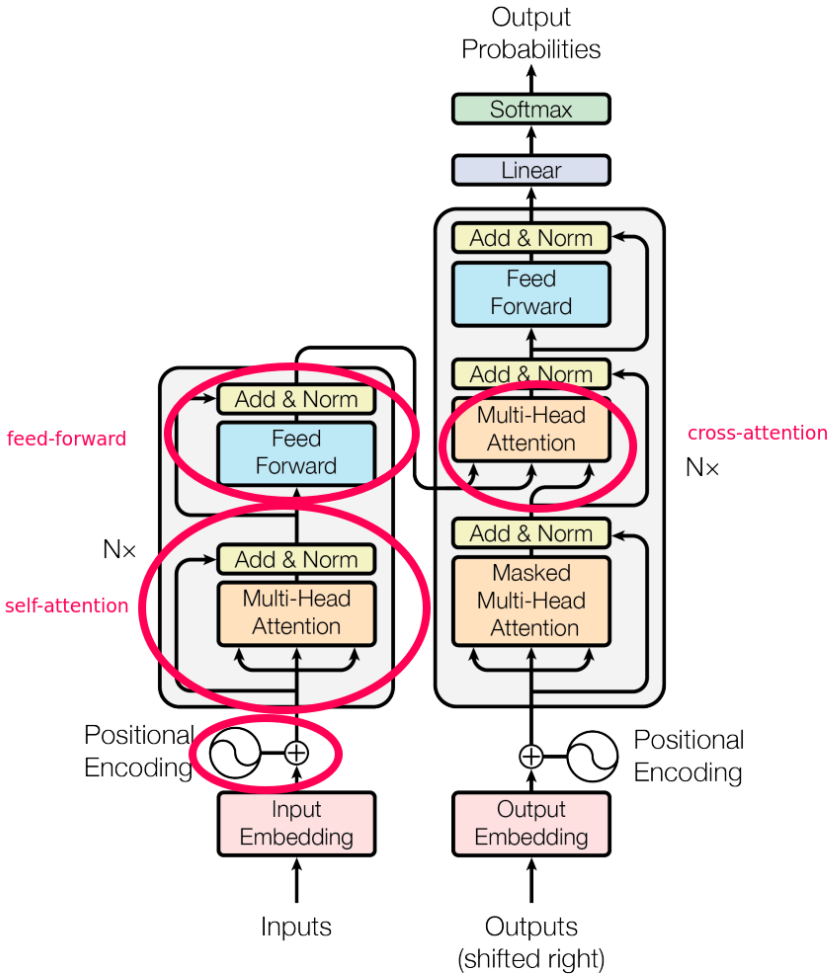
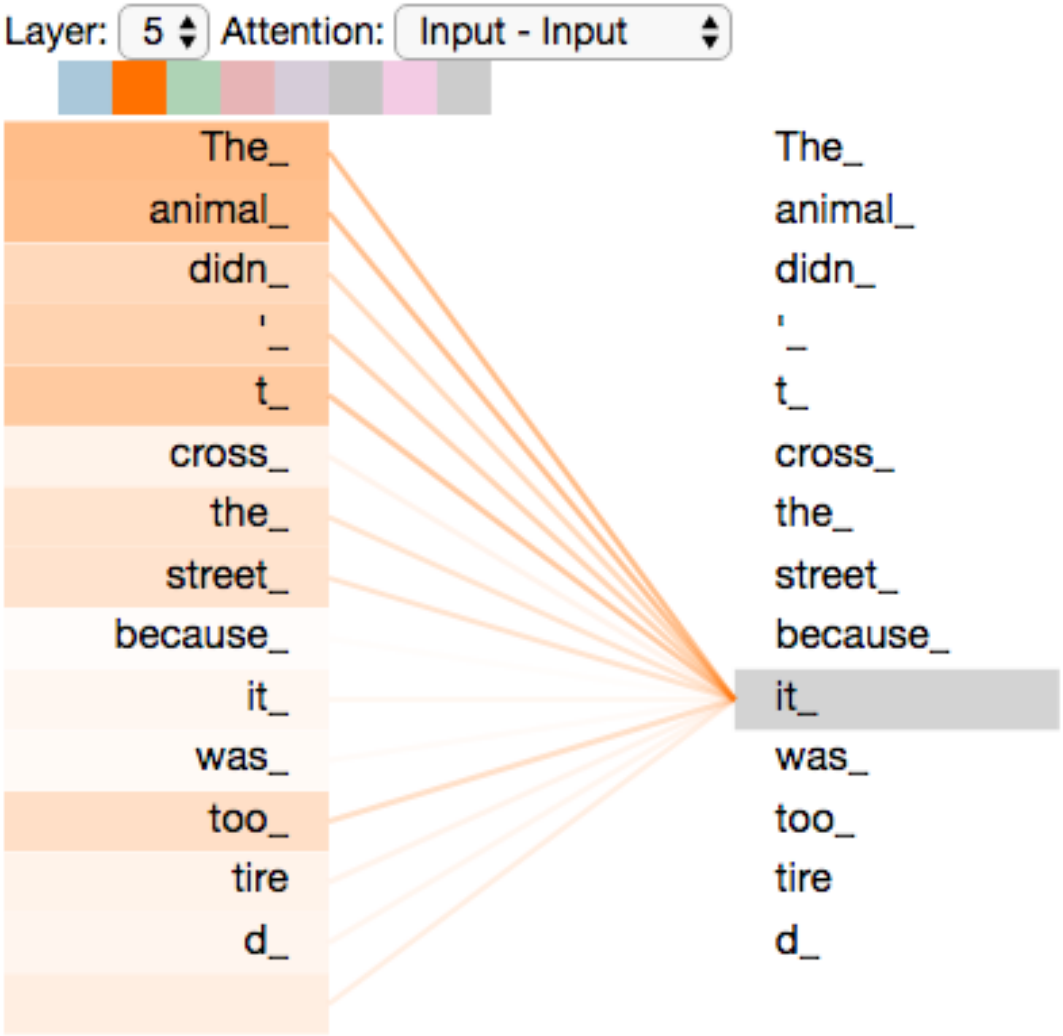


Figure 1: The Transformer - model architecture.

https://paperswithcode.com/methods/category/transformers

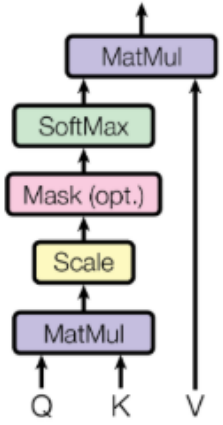
Attention



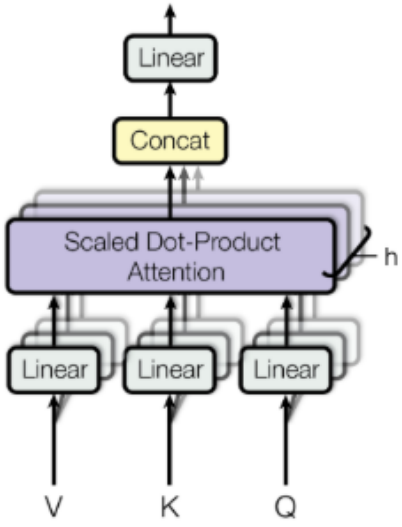
https://paperswithcode.com/methods/category/transformers

Attention

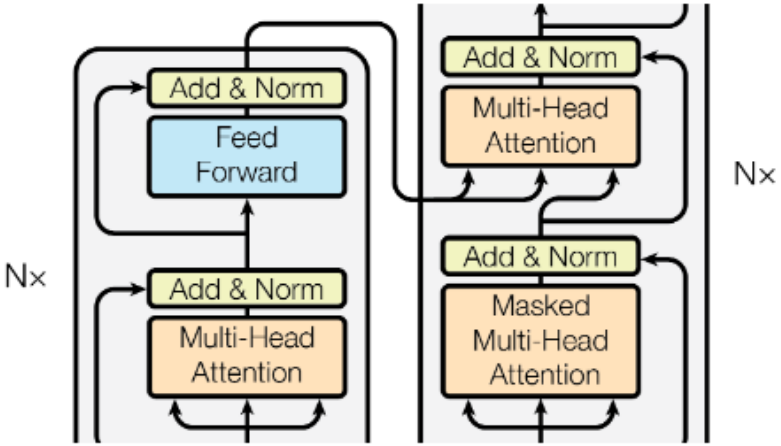
Scaled Dot-Product Attention



Multi-Head Attention

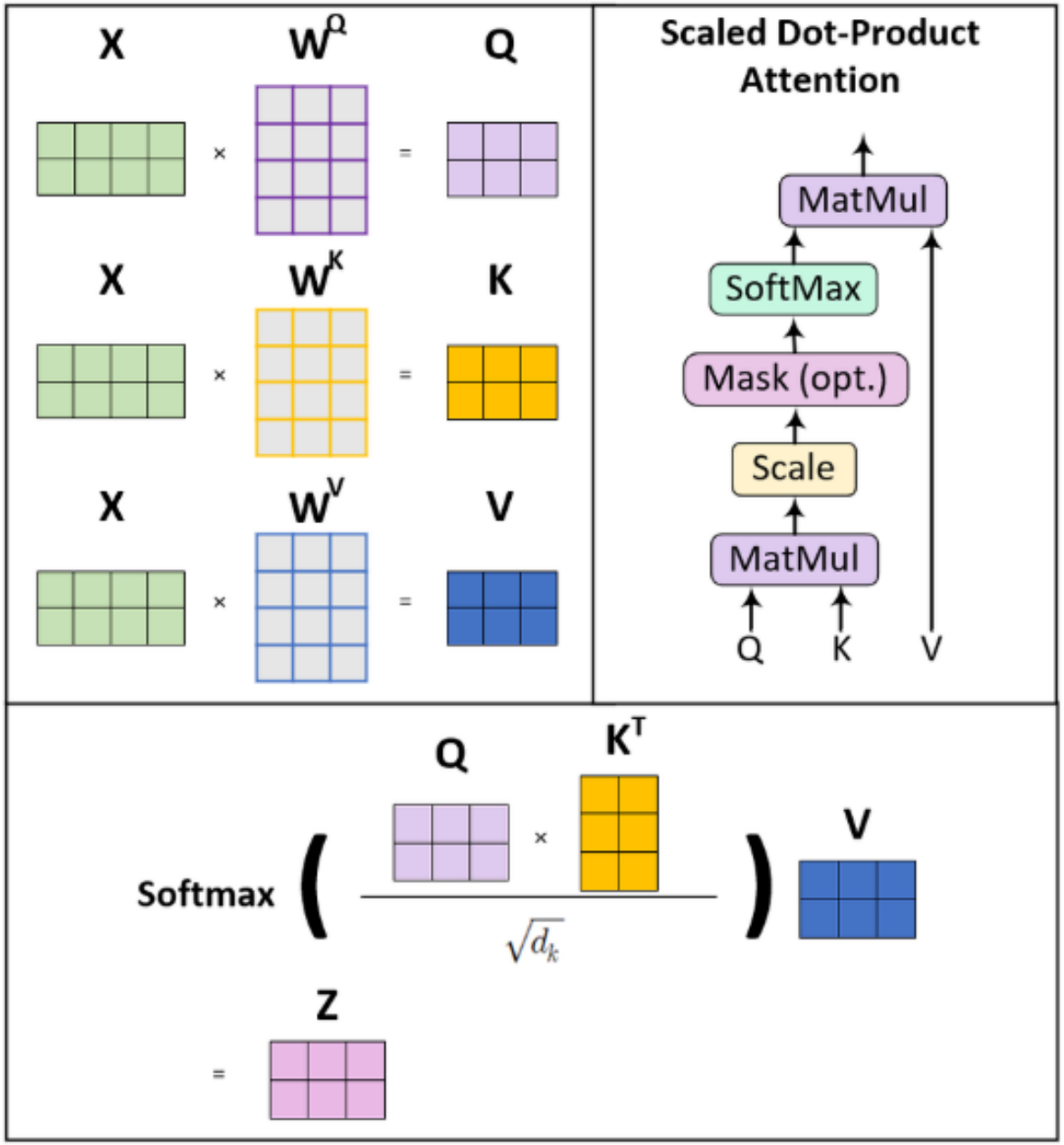


Where in the Transformer model, the Q, K, V values can either come from the same inputs in the encoder (bottom part of the figure below), or from different sources in the decoder (upper right part of the figure). This part is crucial for using this model in translation tasks.



In both papers, as described in the section on attention, the attention values are calculated

Attention



https://www.researchgate.net/figure/Diagram-of-the-query-key-value-and-self-attention-mechanism-The-input-

1. Allgemein: Was sind die Dimensionen bei Attention?

Gegeben sei der Satz

`Die Katze schläft`

mit

$$n = 3$$

Tokens.

Notation

- n: Anzahl der Tokens
- d_{model}: Embedding-Dimension
- d_k: Dimension von Query und Key
- d_v: Dimension von Value

Schritt 1: Embeddings

Neuronale Netze können nicht direkt mit Wörtern rechnen.

Jedes Wort wird daher in einen Zahlenvektor (Embedding) umgewandelt:

$$\texttt{Die} = [0.2,\ 0.8,\ 0.1,\ 0.5]$$

$$\texttt{Katze} = [0.7,\ 0.3,\ 0.9,\ 0.2]$$

$$\texttt{schläft} = [0.1,\ 0.4,\ 0.6,\ 0.8]$$

Die Embedding-Dimension beträgt

$$d_{\text{model}} = 4.$$

Input-Matrix X

Alle Embeddings werden zu einer Matrix zusammengefasst:

$$X = \begin{bmatrix} 0.2 & 0.8 & 0.1 & 0.5 \\ 0.7 & 0.3 & 0.9 & 0.2 \\ 0.1 & 0.4 & 0.6 & 0.8 \end{bmatrix}$$

$$X \in \mathbb{R}^{n \times d_{\text{model}}} = \mathbb{R}^{3 \times 4}$$

Interpretation:

- Zeilen = Tokens
- Spalten = Merkmale des Embeddings

Schritt 2: Query, Key und Value

Jedes Wort wird aus drei Perspektiven betrachtet:

- **Query (Q)**: Wonach suche ich?
- **Key (K)**: Welche Information biete ich an?
- **Value (V)**: Welche Information liefere ich?

Dazu werden drei Gewichtsmatrizen gelernt:

$$W^Q, \qquad W^K, \qquad W^V$$

Dimensionen von Q, K und V

Sei

$$d_k = d_v = 2.$$

Dann gilt

$$W^Q, W^K, W^V \in \mathbb{R}^{4\times 2}.$$

Die Projektionen lauten

$$Q = XW^Q, \qquad K = XW^K, \qquad V = XW^V.$$

Dimension:

$$(3 \times 4)(4 \times 2) = 3 \times 2.$$

Also

$$Q, K, V \in \mathbb{R}^{3\times 2}$$

Dimensionen im Überblick

Matrix	**Dimension**
X	3×4
Q	3×2
K	3×2
V	3×2

Schritt 3: Wer schaut auf wen?

Nun wird berechnet:

$$QK^\top$$

mit

$$Q \in \mathbb{R}^{3\times 2}, \qquad K^\top \in \mathbb{R}^{2\times 3}.$$

Daraus folgt

$$QK^\top \in \mathbb{R}^{3\times 3}.$$

Warum 3×3?

Jedes Wort wird mit jedem anderen Wort verglichen.

Erstes Skalarprodukt

Position $(1, 1)$:

$$q_1^\top k_1 = [0.45, 1.01] \cdot [0.43, 1.00]$$

$$= 0.45 \cdot 0.43 + 1.01 \cdot 1.00 = 1.2035$$

Interpretation:

Wie gut passt Query(Die) zu Key(Die)?

Gesamte Score-Matrix

$$QK^\top = \begin{bmatrix} 1.20 & 1.06 & 1.29 \\ 1.22 & 1.48 & 1.39 \\ 1.17 & 1.21 & 1.28 \end{bmatrix}$$

Jeder Eintrag ist ein Skalarprodukt

$$q_i^\top k_j.$$

Softmax

Die Score-Matrix wird zeilenweise normalisiert:

$$A = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)$$

Beispielsweise wird

$$[1.20,\ 1.06,\ 1.29]$$

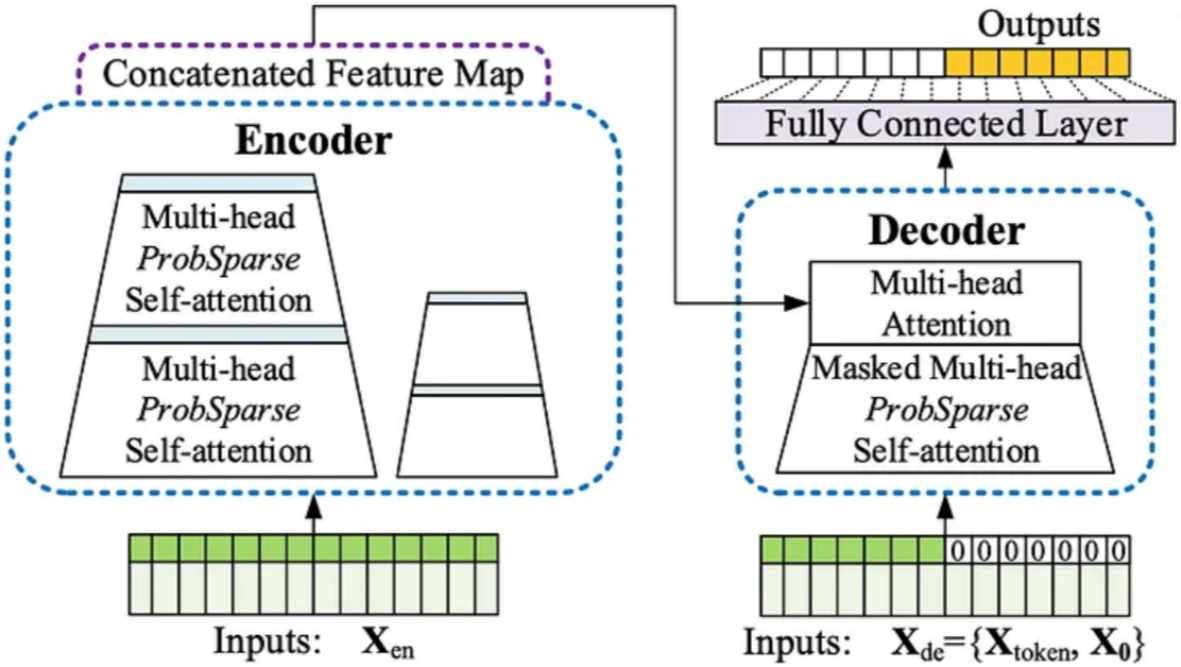
zu

$$[0.34,\ 0.29,\ 0.37].$$

Interpretation:

- 34% Aufmerksamkeit auf sich selbst
- 29% auf `Katze`
- 37% auf `schläft`

Transformer für Zeitreiehen: Informer



Vergleich der Verfahren

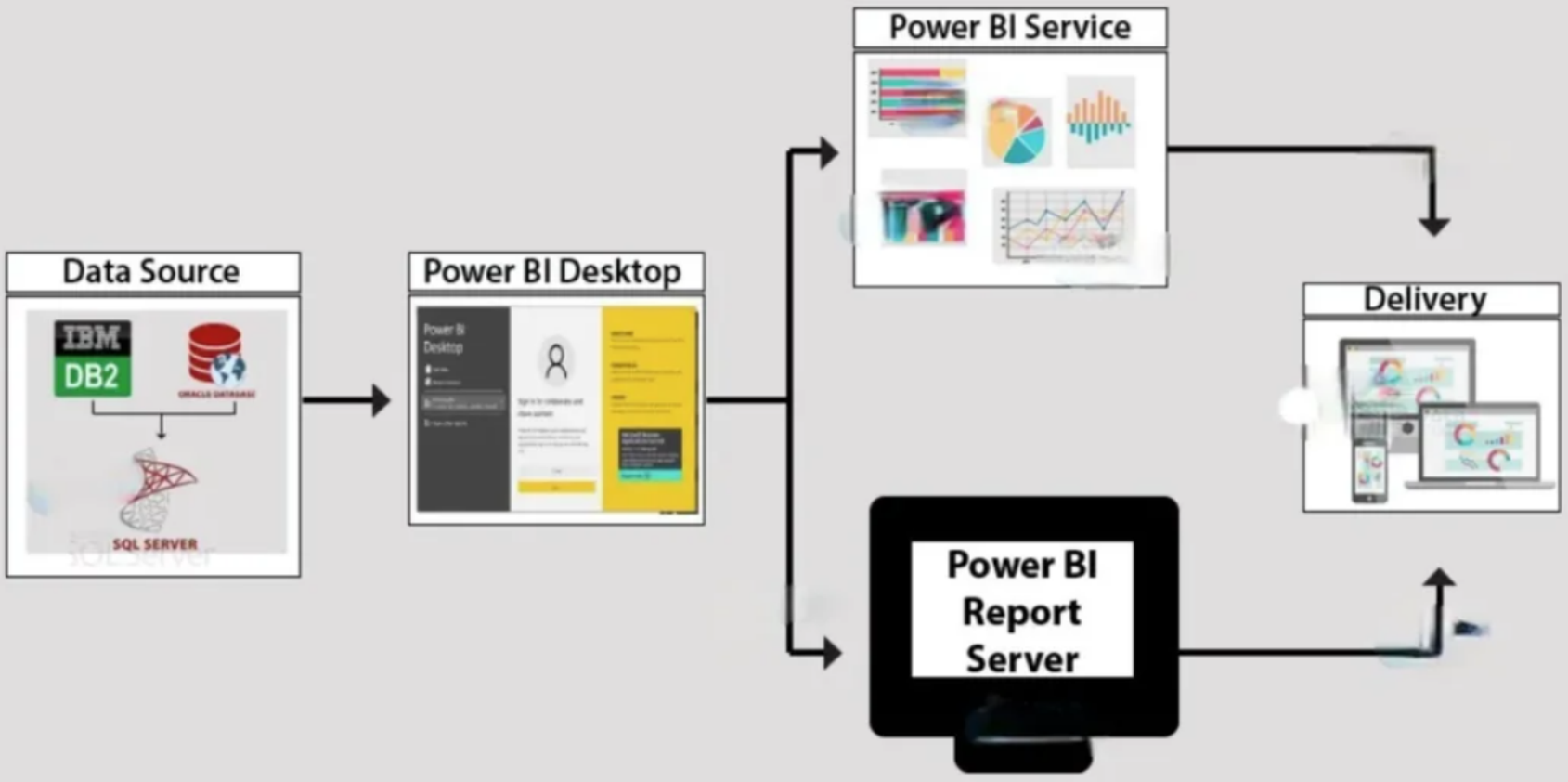
Eigenschaft	RNN	LSTM	Transformer
Kurze Abhängigkeiten	Gut	Gut	Gut
Lange Abhängigkeiten	Schwach	Gut	Sehr gut
Parallelisierung	Nein	Nein	Ja
Trainingsgeschwindigkeit	Mittel	Langsam	Schnell
Prognosegüte	Mittel	Hoch	Sehr hoch

Transformer-Modelle stellen aktuell den Stand der Technik für viele Zeitreihen- und Absatzprognoseaufgaben dar. Sie sind allerdings sehr aufwändig und benötigen viele Daten.

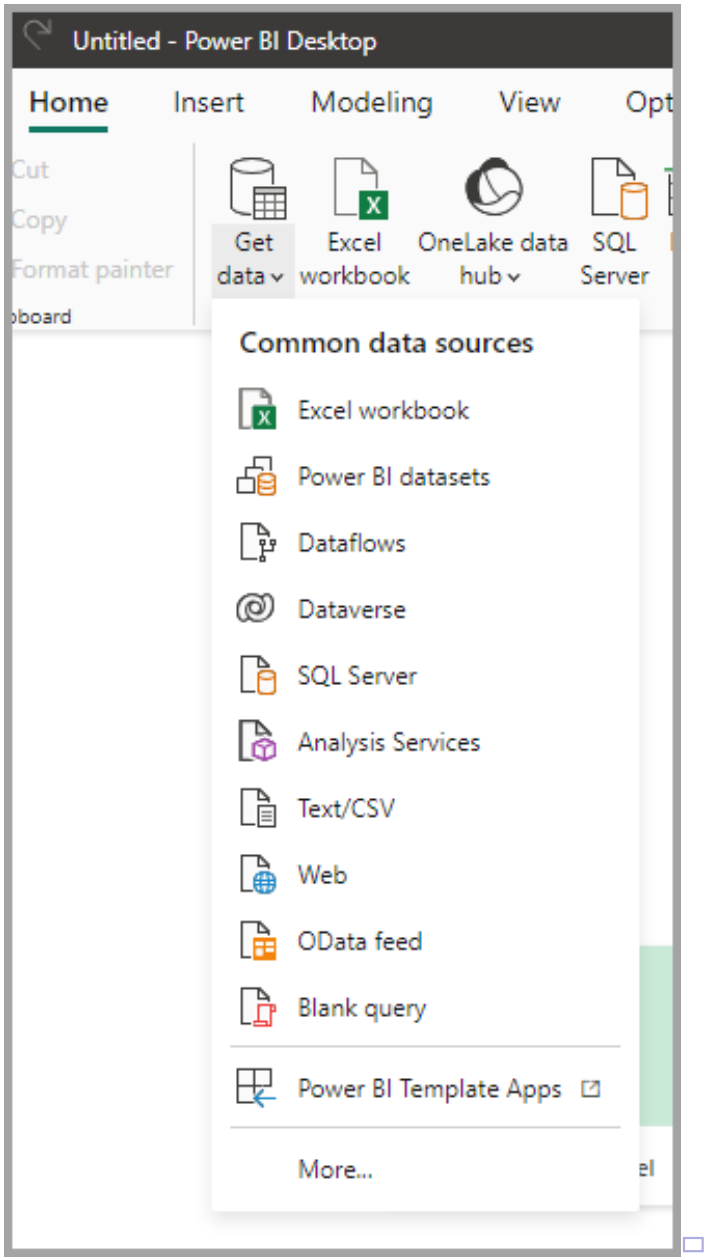
Table Of Contents

1. Absatzprognose
2. Dashboarding

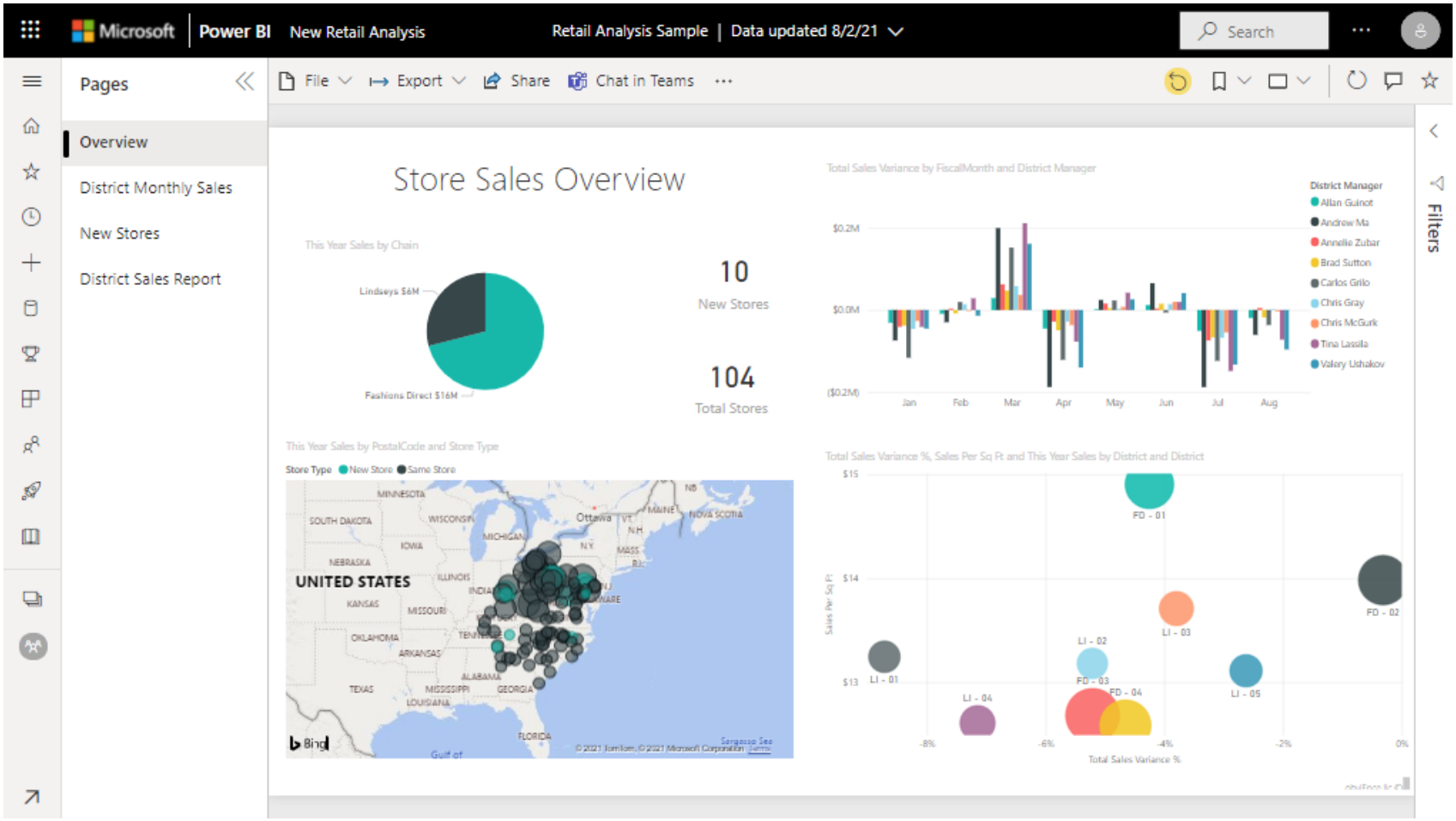
Power BI Architektur



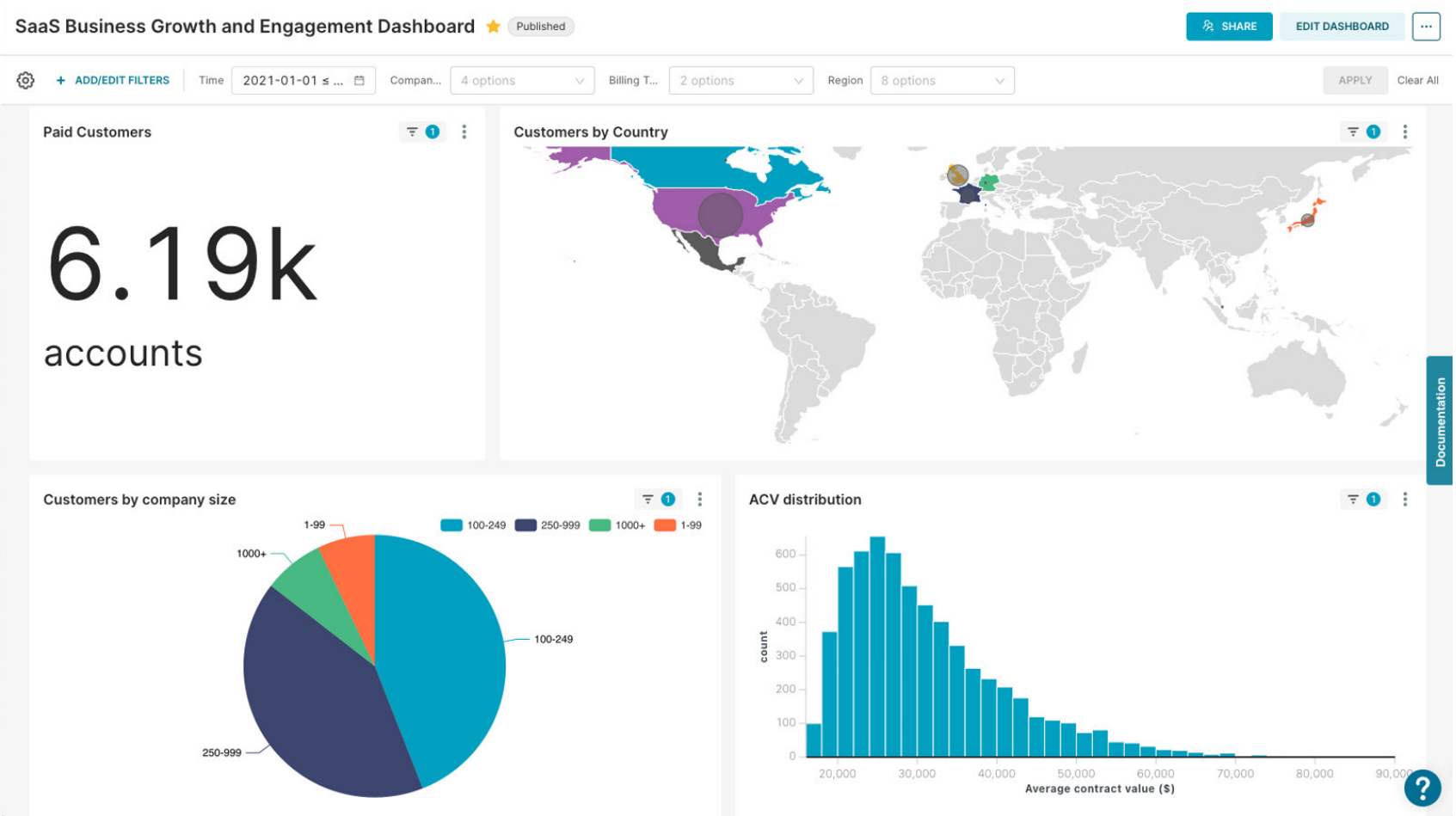
Power BI


Untitled - Power BI Desktop
Home
Insert
Modeling
View
Opt
Cut
Copy
Format painter
Get data
Excel workbook
OneLake data hub
SQL Server
Common data sources
Excel workbook
Power BI datasets
Dataflows
Dataverse
SQL Server
Analysis Services
Text/CSV
Web
OData feed
Blank query
Power BI Template Apps
More...

Power BI


Microsoft
Power BI
New Retail Analysis
Retail Analysis Sample | Data updated 8/2/21
Search
Pages
Overview
District Monthly Sales
New Stores
District Sales Report
File
Export
Share
Chat in Teams
Store Sales Overview
This Year Sales by Chain
Lindseys $6M
Fashions Direct $16M
10
New Stores
104
Total Stores
Total Sales Variance by FiscalMonth and District Manager
District Manager
Allan Guinot
Andrew Ma
Annelie Zubar
Brad Sutton
Carlos Grilo
Chris Gray
Chris McGurk
Tina Lassila
Valery Ushakov
$0.2M
$0.0M
($0.2M)
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Filters
This Year Sales by PostalCode and Store Type
Store Type
New Store
Same Store
UNITED STATES
Total Sales Variance %, Sales Per Sq Ft and This Year Sales by District and District
$15
$14
$13
Sales Per Sq Ft
FD - 01
FD - 02
LI - 01
LI - 02
LI - 03
LI - 04
LI - 05
FD - 03
FD - 04
-8%
-6%
-4%
-2%
0%
Total Sales Variance %

Superset


SaaS Business Growth and Engagement Dashboard
Published
SHARE
EDIT DASHBOARD
+ ADD/EDIT FILTERS
Time
2021-01-01 ≤ ...
Compan...
4 options
Billing T...
2 options
Region
8 options
APPLY
Clear All
Paid Customers
6.19k
accounts
Customers by Country
Documentation
Customers by company size
1-99
1000+
100-249
250-999
ACV distribution
count
Average contract value ($)