Machine Learning

Prof. Dr. Fabian Brunner

<fa.brunner@oth-aw.de>

Amberg, 28. November 2023

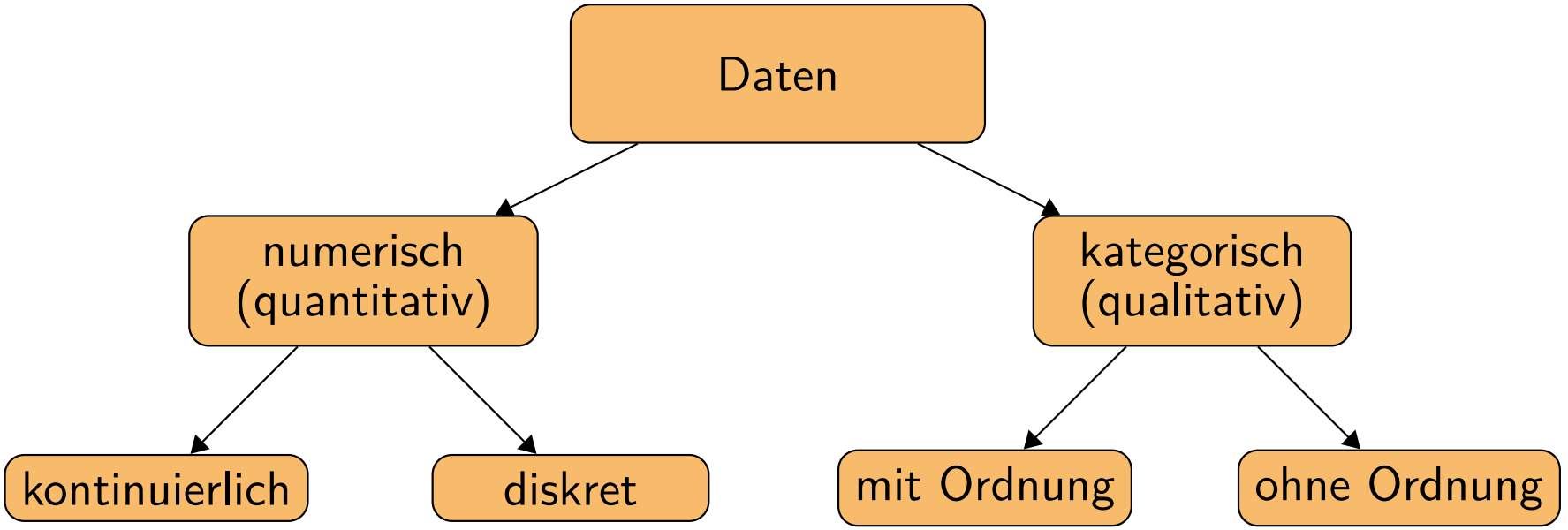
Übersicht

Ostbayerische Technische Hochschule
Amberg-Weiden

Thema heute: Datenvorbereitung

- Sampling
- Behandlung fehlender Werte
- Kodierung
- Standardisierung und Normierung
- Binning
- ColumnTransformers und Pipelines in Scikit-learn
- Übung: Datenvorbereitung mit Pandas
- Übung: Transformers in Scikit-learn
- Übung: Pipelines in Scikit-learn

Datentypen



z.B. Gewicht

z.B. Anzahl Kinder

z.B. Gehaltsstufe

z.B. Geschlecht

Skalenniveaus

- Sorte: Gala — kategorisch, ohne Ordnung, nominal
- Zuckergehalt: hoch — ordinal, kat. mit O
- Ernte: 25.09.2019 — numerisch, Intervall
- Gewicht: 120g — Verhältnis

Skalenniveaus

Worin unterscheiden sich die Merkmale?

Sorte	Zuckergehalt	Ernte	Gewicht
Gala	hoch	25.09.2019	120g
Braeburn	mittel	13.10.2019	230g
Boskop	gering	10.10.2019	190g
Granny Smith	gering	15.09.2019	200g

Skalenniveaus

- **Nominalskala:** Ausprägungen können unterschieden werden.
- **Ordinalskala:** Es besteht zusätzlich eine Rangordnung/Reihenfolge
- **Intervallskala:** Es können zusätzlich Differenzen gebildet und Abstände gemessen werden
- **Verhältnisskala:** Es können Verhältnisse gebildet werden.

Skalenniveaus - Zusammenfassung

Skalenniveau		Operationen	Messbare Eigenschaften	Beispiel
Nominalskala		$=, \neq$	Häufigkeit	Apfelsorte
Ordinalskala		$=, \neq, >, <$	Häufigkeit Rangfolge	Zuckergehalt
Kardinalskala	Intervallskala	$=, \neq, >, <$ $+, -$	Häufigkeit Rangfolge Abstand	Datum
	Verhältnisskala	$=, \neq, >, <$ $+, -, \div$	Häufigkeit Rangfolge Abstand Verhältnis	Gewicht

Datenvorbereitung

Ostbayerische Technische Hochschule Amberg-Weiden

Die Daten für ein Machine-Learning-Problem können unterschiedlichen Quellen stammen und u.U. heterogen sein. Ein wichtiger Schritt vor der eigentlichen Modellierung ist die Datenvorbereitung. Dazu zählen beispielsweise folgende Aufgaben:

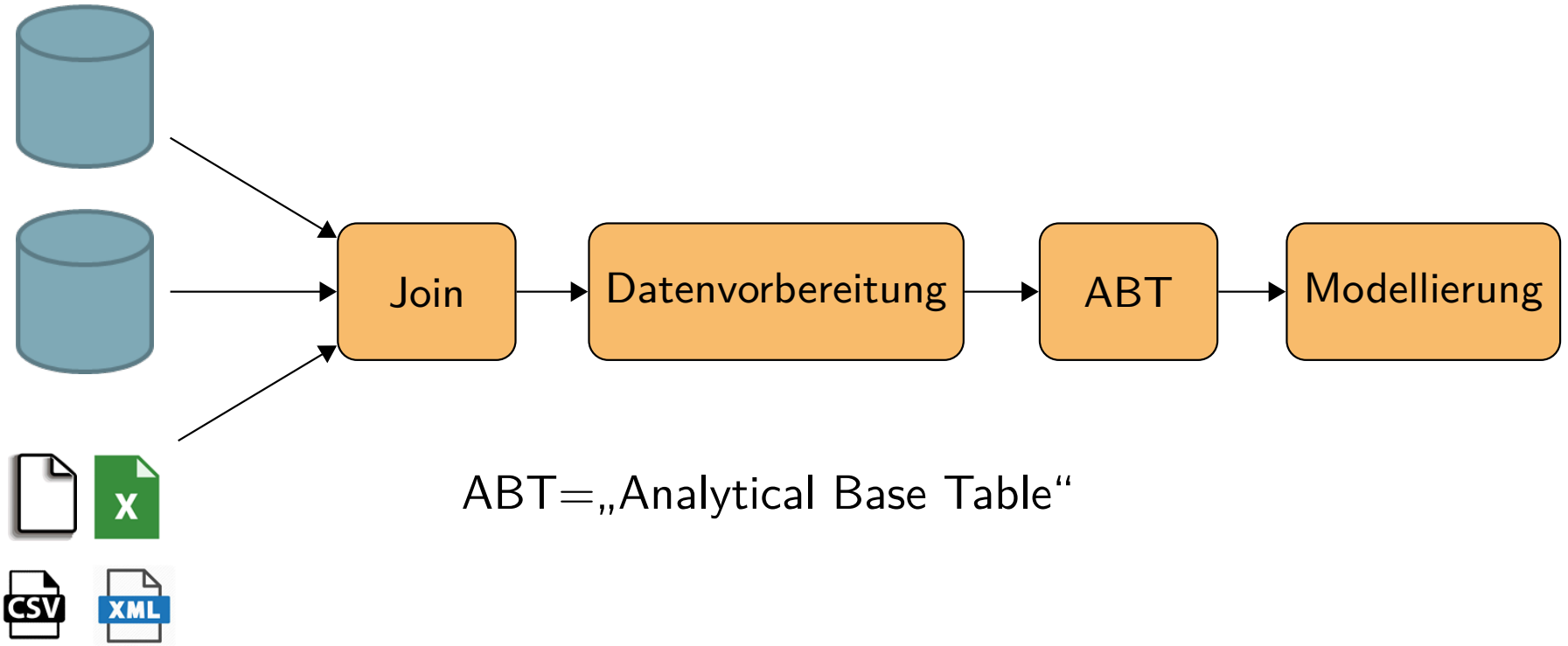
- Datenakquise und -zusammenführung
- Harmonisierung
- Feature Engineering
- ggf. Anonymisierung
- Filterung, Sampling
- Bereinigung (fehlende Werte, Ausreißer, Fehlerbehebung, Behandlung von Rauschen)
- Transformation (ABT, Strukturierung, Klassenbildung, Kodierung, Anpassung von Formaten, Skalierung, Standardisierung etc.)

Strukturierte und unstrukturierte Daten

Kategorie	Definition	Beispiel
Strukturierte Daten	Daten, die einer formalisierten Struktur (z.B. Tabellenform) in bestimmten Formaten und Datentypen vorliegen; können durch Computerprogramme einfach und effizient abgefragt, aggregiert und verarbeitet werden können.	Relationale Datenbanken, Excel-Tabellen, csv-Dateien
Semistrukturierte Daten	Daten, die nicht in der formalisierten Struktur einer relationalen Datenbank vorliegen, jedoch Schlüssel oder Strukturierungselemente enthalten, die eine Daten-Hierarchie vorgeben und durch die semantische Elemente separiert werden.	JSON, E-Mail, XML
Unstrukturierte Daten	Daten, die in keiner formalisierten Struktur vorliegen.	Freitext

Datenzusammenführung

- Zusammenführung von Daten aus unterschiedlichen, ggf. heterogenen, Quellen in einer einheitlichen Ziel-Datenstruktur.
- ggf. Generierung neuer Merkmale durch Aggregation
- Beispiele für Datenquellen: Relationale Datenbanken, Excel-Dateien, CSV-Dateien, NoSQL-Datenbanken, Data Streams, Web-APIs, etc.

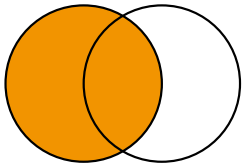


ABT=„Analytical Base Table“

Datenzusammenführung und -aggregation

Tabelle „Kunden“

ID	Name	Vorname
1	Klein	Barbara
2	Berger	Thomas
3	Müller	Ina
4	Welsch	Kevin



Left join

ID	Name	Vorname	Umsatz
1	Klein	Barbara	55,00
2	Berger	Thomas	33,00
3	Müller	Ina	50,00
4	Welsch	Kevin	NaN

Tabelle „Umsätze“

ID	Datum	Umsatz
1	01.02.19	10,00
1	12.04.19	10,00
1	14.10.19	35,00
2	19.05.19	21,00
2	24.09.19	12,00
3	19.08.19	50,00

Bei der Datenzusammenführung entstehen häufig fehlende Werte

Modellierungsdatensatz

ABT (Analytical Base Table)

- Eingabedatensatz für ein ML-Modell
- Viele Verfahren benötigen vollbesetzten Datensatz ohne Missings
- Viele Verfahren benötigen ausschließlich numerische Werte

$\longrightarrow$ **Datenbereinigung und -transformation**

Feature 1	...	Feature p	Zielvariable
195	...	0	0
149	...	0	1
223	...	1	1
132	...	0	0
279	...	0	1
184	...	1	1
⋮		⋮	⋮

m Trainings-datensätze

Feature Engineering

=> je nach Fachbereich

Ostbayerische Technische Hochschule Amberg-Weiden

- Oftmals kann man aus den verfügbaren Daten weitere, aussagekräftige Features ableiten/konstruieren.
- Dies kann bereits bei der Datenvorbereitung geschehen, oder im Rahmen der Modellierung, um ein bestehendes Modell (iterativ) zu verbessern.
- Feature Engineering ist eine zentrale Aufgabe und erfordert typischerweise domänenspezifisches Wissen.
- Später werden wir noch lernen, wie man im Zuge der Dimensionsreduktion durch PCA (Hauptkomponentenanalyse, **p**rincipal **c**omponent **a**nalysis) durch Transformation neue Features generieren kann.
- Beispiel: Gewicht und Körpergröße $\rightarrow$ BMI
- Weitere Aufgabe bei der Modellierung: Feature Selection

train_test_split danach standardisieren, weil sonst Teile der Train daten in Testdaten einfließen

=> Auswahl von Features

Sampling-Strategien

Simple random sampling

- Auswahl einer Zufallsstichprobe
- gleiche Wahrscheinlichkeit für jedes Subsample der Länge k, in die Stichprobe aufgenommen zu werden

Systematic sampling

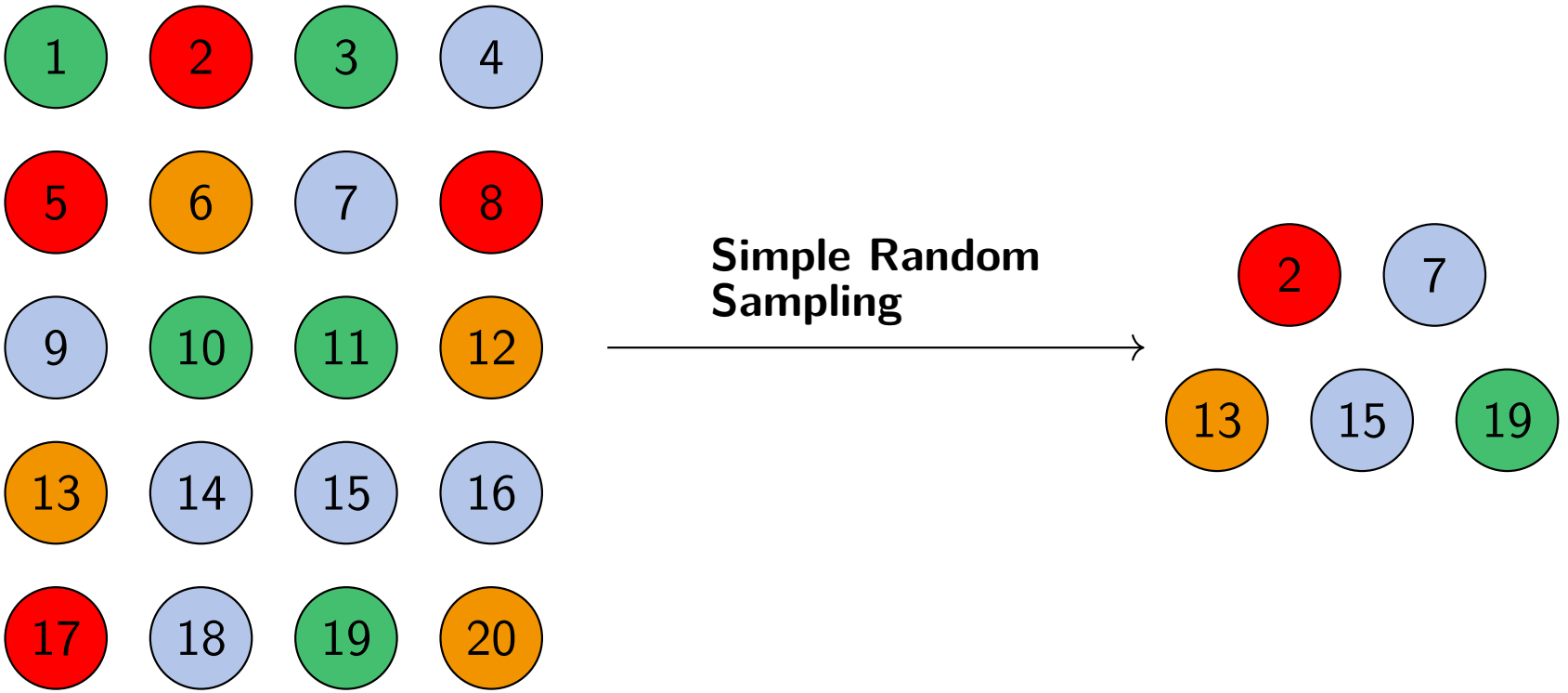
- Wähle zufälliges Element aus. Gehe danach den Datensatz (zyklisch) durch und selektiere jedes k-te Element.
- nur bei homogen verteilten Daten sinnvoll

Stratified Sampling

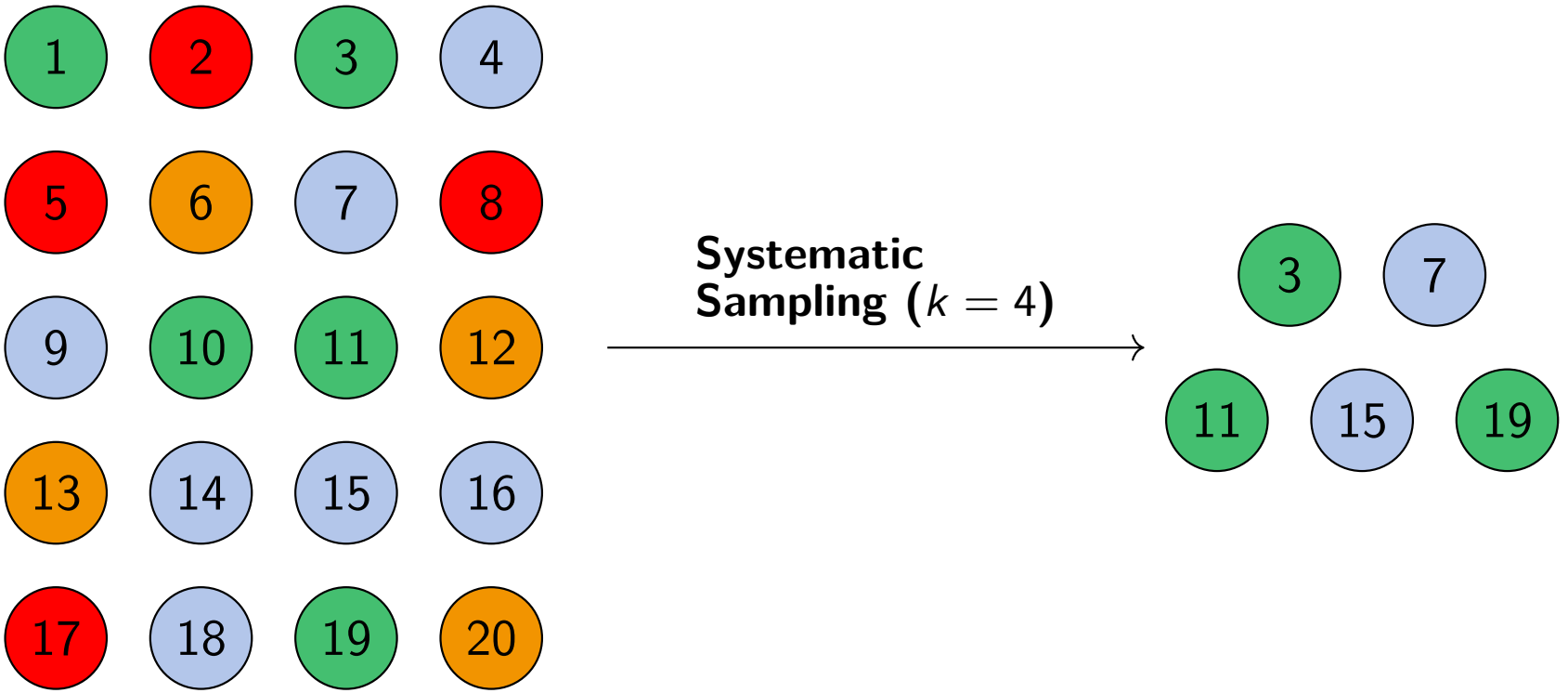
- Ziehen einer Stichprobe aus einer Datenmenge, die in Teilmengen aufgeteilt werden kann
- Beispiel: Datensätze mit verschiedenen (diskreten) Zielfunktionswerten
- Mögliches Ziel: Kontrolle der Anteile der Zielfunktionswerte in der Stichprobe (z.B. bei unbalancierten Datensätzen)

Sampling mit Pandas: `pandas.DataFrame.sample`

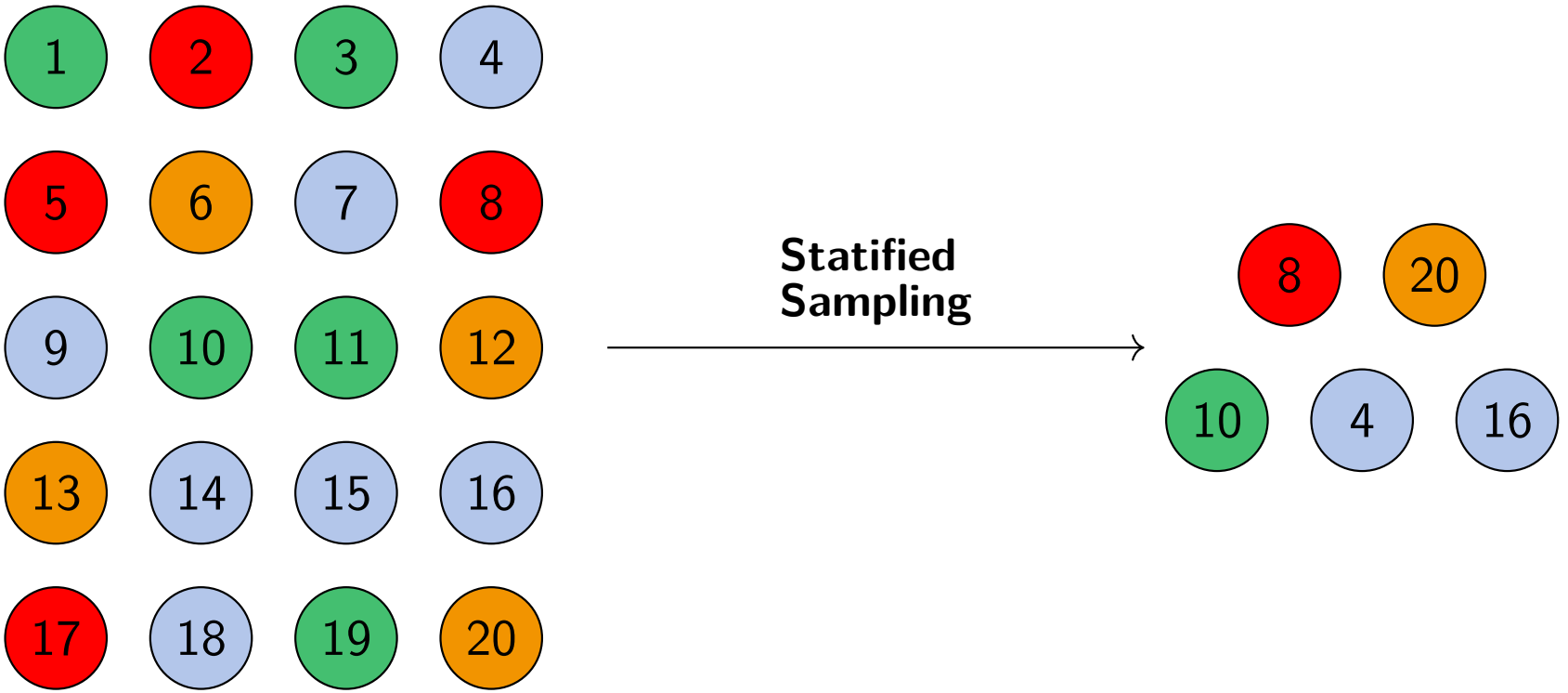
Beispiel zu Sampling-Strategien



Beispiel zu Sampling-Strategien



Beispiel zu Sampling-Strategien



Imputation: fehlende Werte behandeln

Ostbayerische Technische Hochschule Amberg-Weiden

- Die meisten Machine Learning-Modelle erwarten vollbesetzte Datensätze.
- Datensätze mit fehlenden Daten müssen weggelassen werden, oder die fehlenden Werte müssen ersetzt werden. Diesen Vorgang nennt man **Imputation**.
- Mögliche Strategien: ersetze durch häufigsten Wert (bei kategorischen Variablen) oder durch einen Dummy-Wert (z.B. den Mittelwert der Daten).
- Häufig werden binäre Indikatorspalten eingeführt, um die Information, dass Missings vorlagen, nicht zu verlieren.
- Die Dummy-Werte müssen anhand des Trainingsdatensatzes berechnet werden.
- Häufiger Fehler: separate Berechnung von Mittelwerten auf dem Testdatensatz und Verwendung dieser Mittelwerte.
- Imputation in Pandas: `DataFrame.fillna`
- Imputation in Scikit-learn: Klasse `sklearn.impute.SimpleImputer`
- Hinzufügen von Indikatorspalten: Klasse `sklearn.impute.MissingIndicator`

Imputation in Pandas

```
df = pd.DataFrame({"age":[35,55,np.nan,30],
        "salary": [3600, np.nan, 2800, 4100]})
# Ausgabe von df:
#     age  salary
# 0  35.0  3600.0
# 1  55.0     NaN
# 2   NaN  2800.0
# 3  30.0  4100.0
column_means = df.mean()
df_imp = df.apply(lambda x: x.fillna(column_means[x.name]))
df_imp
#  Ausgabe:
#     age  salary
# 0  35.0  3600.0
# 1  55.0  3500.0
# 2  40.0  2800.0
# 3  30.0  4100.0
```

Imputation in Scikit-learn

```
from sklearn.impute import SimpleImputer

df = pd.DataFrame({"age":[35,55,np.nan,30],
"salary": [3600, np.nan, 2800, 4100]})

im = SimpleImputer(missing_values=np.nan, strategy='mean')
im.fit(df)
im.transform(df)

# Ausgabe
# array([[  35., 3600.],
#        [  55., 3500.],
#        [  40., 2800.],
#        [  30., 4100.]])
```

Label Encoding

- Umwandlung von kategorischen Daten in numerische Werte
- Jeder Ausprägung eines Features wird ein Integer-Zahlenwert zugewiesen.
- Problem: es entsteht eine (künstliche) Ordnung zwischen den Ausprägungen. Daher wird Label Encoding üblicherweise nur für die Kodierung der Zielvariable verwendet.

Land
GER
IT
CH
FR
GER
CH
CH
GER
FR

Label Encoding →

Land_{LE}
1
2
3
4
1
3
3
1
4

Label Encoding in Scikit-learn

- Mit dem LabelEncoder in Scikit-learn kann eine kategorische Zielvariable numerisch kodiert werden.
- Er implementiert die Methoden `fit` und `transform`.
- Mittels `fit` kann man die Ausprägungen übergeben, die kodiert werden sollen.
- Die Methode `transform` führt die Kodierung durch.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

y = ["no", "no", "yes", "no"]
le = LabelEncoder()
le.fit(["yes","no"])
le.transform(y)

#Ausgabe: array([0, 0, 1, 0], dtype=int64)
```

One Hot Encoding

- Führe pro Ausprägung eines kategorischen Merkmals eine binäre Spalte ein
- Durch One Hot Encoding enstehen kollineare Spalten. Mindestens eine kann weggelassen werden.
- Variante: kodiere nur die häufigsten n Ausprägungen und führe eine zusätzliche "Rest-Spalte" ein.
- One-Hot-Encoding in Pandas: *pandas.DataFrame.get_dummies*

Land
GER
IT
CH
FR
GER
CH
CH
GER
FR

One Hot Encoding →

Land_{GER}	Land_{IT}	Land_{CH}	Land_{FR}
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0
0	0	1	0
0	0	1	0
1	0	0	0
0	0	0	1

One Hot Encoding in Pandas

Verwendung der Pandas-Funktion `get_dummies`
Setzt man `drop_first=True`, so wird die erste kodierte Spalte (die linear abhängig zu den übrigen ist) weggelassen.

```
import pandas as pd

X = pd.DataFrame({"country":["GER","CH","FR","GER"],
 "gender":["m", "f", "m", "m"]})
pd.get_dummies(X)

#Ausgabe
#    country_CH  country_FR  country_GER  gender_f  gender_m
#0            0           0            1         0         1
#1            1           0            0         1         0
#2            0           1            0         0         1
#3            0           0            1         0         1
```

One Hot Encoding in Scikit-learn

Ostbayerische Technische Hochschule Amberg-Weiden

Alternativ zur Pandas-Methode `get_dummies` kann der in Scikit-learn verfügbare One Hot Encoder verwendet werden. Beispiel:

```
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(handle_unknown='ignore', sparse=False)

X = pd.DataFrame({"country":["GER","CH","FR","GER"],
 "gender":["m", "f", "m", "m"]})
ohe.fit_transform(X)
# Ausgabe:
# array([[0., 0., 1., 0., 1.],
#        [1., 0., 0., 1., 0.],
#        [0., 1., 0., 0., 1.],
#        [0., 0., 1., 0., 1.]])
```

Tipp: mit der Methode `get_feature_names_out` kann man sich die Namen der transformierten Spalten ausgeben lassen.

Diskretisierung durch Binning

- Manchmal möchte man kontinuierliche Werte diskretisieren und in Klassen einteilen (z.B. Altersgruppen, Angabe der Uhrzeit in vollen Stunden etc.).
- In Pandas kann dieses „Binning“ mit Hilfe der Funktionen cut und qcut durchgeführt werden.

```
import numpy as np
df = pd.DataFrame({'age': np.random.randint(21, 41,5 )})
df['age_bins'] = pd.cut(x=df['age'], bins=[20, 30, 40])
df['age_by_decade'] = pd.cut(x=df['age'], bins=[20, 30, 40],
labels=['20s', '30s'])
#Ausgabe:
#    age  age_bins age_by_decade
# 0   26  (20, 30]           20s
# 1   39  (30, 40]           30s
# 2   31  (30, 40]           30s
# 3   38  (30, 40]           30s
# 4   21  (20, 30]           20s
```

Normierung

- Liegen kontinuierliche Features vor, deren Größenordnungen sich stark unterscheiden, so kann dies die Konvergenzgeschwindigkeit von Optimierungs-Verfahren, die zum Modelltraining eingesetzt werden, negativ beeinflussen (z.B. verzerrte Niveaulinien beim Least-Squares-Funktional, vgl. VL über Lineare Regression).
- Im Rahmen der Vorverarbeitung sollten solche Merkmale geeignet transformiert werden.
- Durch Normierung kann der Wertebereich eines Merkmals auf ein vorgegebenes Intervall, z.B. $[0, 1]$, abgebildet werden.
- Beispiel: Min-Max-Skalierung

Min-Max-Skalierung

Sei $\boldsymbol{x}^{(i)}$ das i-te Feature eines Datensatzes und sei $x_{min}^{(i)}$ dessen Minimum und $x_{max} \neq x_{min}^{(i)}$ dessen Maximum. Dann ist das durch Min-Max-Skalierung transformierte Feature definiert durch

$$\boldsymbol{x}_{norm}^{(i)} := \frac{\boldsymbol{x}^{(i)} - x_{min}^{(i)}}{x_{max}^{(i)} - x_{min}^{(i)}} \ .$$

Standardisierung

- Min-Max-Skalierung ist empfindlich gegenüber Ausreißern. Betrachte zum Beispiel $\boldsymbol{x} = (1, 1.01, -0.9, 1.0, 1.0, \ldots, 1.0, 10000)^T$.
- Bei der Standardisierung wird ein Feature mit arithmetischem Mittel 0 und empirischer Standardabweichung 1 wie folgt erzeugt:

Standardisierung

Sei x ein Feature, das im Datensatz durch die Beobachtungen $\boldsymbol{x} = (x^{(1)}, \ldots x^{(m)})^T$ repräsentiert wird. Dann geht die standardisierter Größe $\hat{x}$ aus x durch Subraktion des empirischen Mittelwerts und Division der empirischen Standardabweichung hervor:

$$\hat{\boldsymbol{x}} := \frac{\boldsymbol{x} - \bar{\boldsymbol{x}}}{s} \ ,$$

wobei

$$\bar{\boldsymbol{x}} = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} \ , \quad s = \sqrt{\frac{1}{m-1} \sum_{i=1}^{m} (x^{(i)} - \bar{\boldsymbol{x}})^2} \ .$$

Standardisierung in Scikit-learn

```
from sklearn.preprocessing import StandardScaler
df = pd.DataFrame({"val1":[1.1,2.2,10.5,-9.3],
        "val2": [1,1,2,2]})
sc = StandardScaler()
sc.fit(df)
sc.transform(df)

# array([[-0.00355579, -1.        ],
#        [ 0.1528991 , -1.        ],
#        [ 1.33342236,  1.        ],
#        [-1.48276567,  1.        ]])

sc.transform(df).mean(axis=0)
# array([-5.55111512e-17,  0.00000000e+00])

sc.transform(df).std(axis=0)
# array([1., 1.])
```

Normierung in Scikit-learn

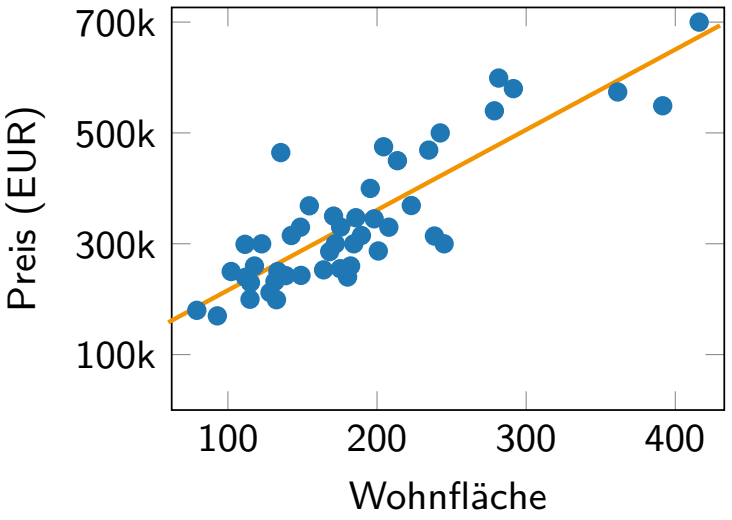
```
from sklearn.preprocessing import MinMaxScaler

df = pd.DataFrame({"val1":[1.1,2.2,10.5,-9.3],
        "val2": [1,1,2,2]})

mmsc = MinMaxScaler()
mmsc.fit(df)
mmsc.transform(df)

# Ausgabe
# array([[0.52525253, 0.        ],
#        [0.58080808, 0.        ],
#        [1.        , 1.        ],
#        [0.        , 1.        ]])
```

Standardisierung des Häuserpreis-Datensatzes

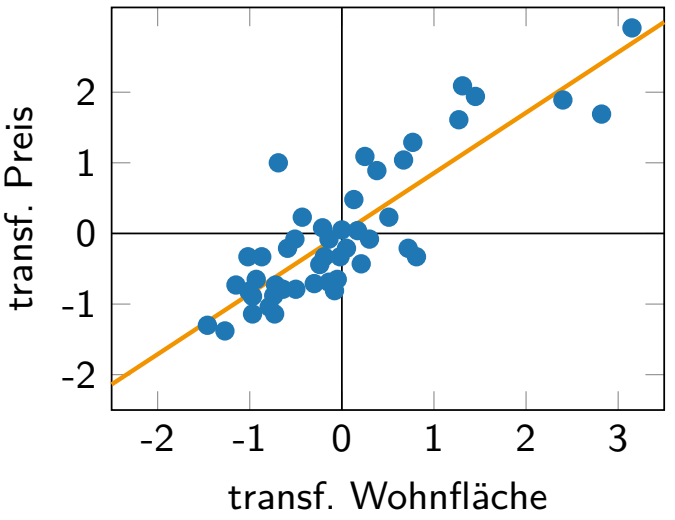
- Standardisierung erweist sich als nützlich, wenn iterative Optimierungsmethoden zum Modell-Fitting eingesetzt werden. Diese werden häufig mit 0 oder Zufallswerten nahe 0 initialisiert.
- Mit Hilfe der Standardisierten lassen sich Ausreißer erkennen.

Beispiel: Häuserpreise-Datensatz (vgl. VL über Lineare Regression)

Unskaliert



Standardisiert



Scikit-learn Estimator API

Ostbayerische Technische Hochschule Amberg-Weiden

Estimators in Scikit-learn

- Die Basisklasse für Machine-Learning-Modelle in Scikit-learn ist die Klasse `sklearn.base.BaseEstimator`.
- Möchte man eigene Modelle entwickeln, kann man von dieser Klasse ableiten.
- Estimators implementieren typischerweise die folgenden Methoden:

`__init__`	Konstruktor
`fit`	Modell-Fitting
`predict`	Modell-Anwendung
`score`	Berechnung eines Gütemaßes ($\rightarrow$ GridSearch)

- Jeder Estimator kann am Ende einer Pipeline (s. gleich) stehen.
- Basisklasse für Klassifikatoren: `sklearn.base.ClassifierMixin`
- Basisklasse für Regressionsmodelle: `sklearn.base.RegressionMixin`

Scikit-learn Transformer API

Ostbayerische Technische Hochschule Amberg-Weiden

Transformer in Scikit-learn

- Die Basisklasse für Transformer in Scikit-learn ist `sklearn.base.TransformerMixin`
- Möchte man eigene Transformer implementieren, kann man von dieser Klasse ableiten.
- Ein Transformer stellt die folgenden Methoden bereit:

`__init__`	Konstruktor
`fit`	Parameter-Schätzung
`transform`	Durchführung der Transformation

- Wird er von `TransformerMixin` abgeleitet, so wird zusätzlich die Methode `fit_transform` hinzugefügt, die beide Schritte hintereinander ausführt.
- Implementierung eines eigenen Transformers: siehe Übung

ColumnTransformer in Scikit-learn

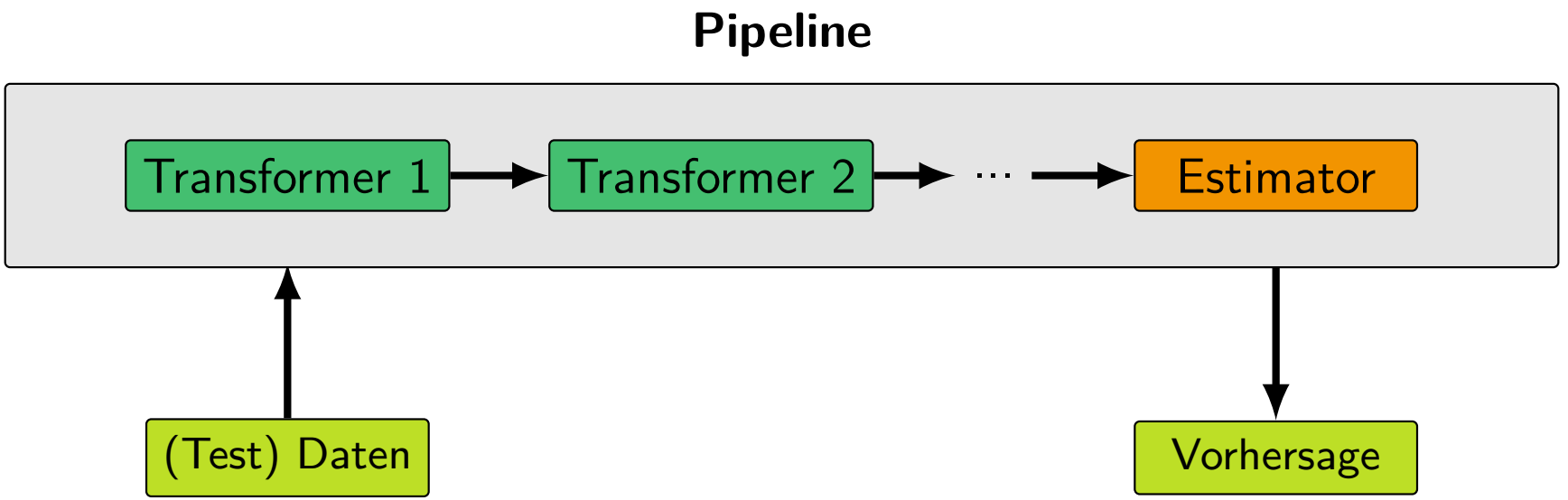
Ostbayerische Technische Hochschule Amberg-Weiden

Mit einem `ColumnTransformer` können unterschiedliche Vorverarbeitungsschritte gleichzeitig auf einzelne Spalten angewendet werden:

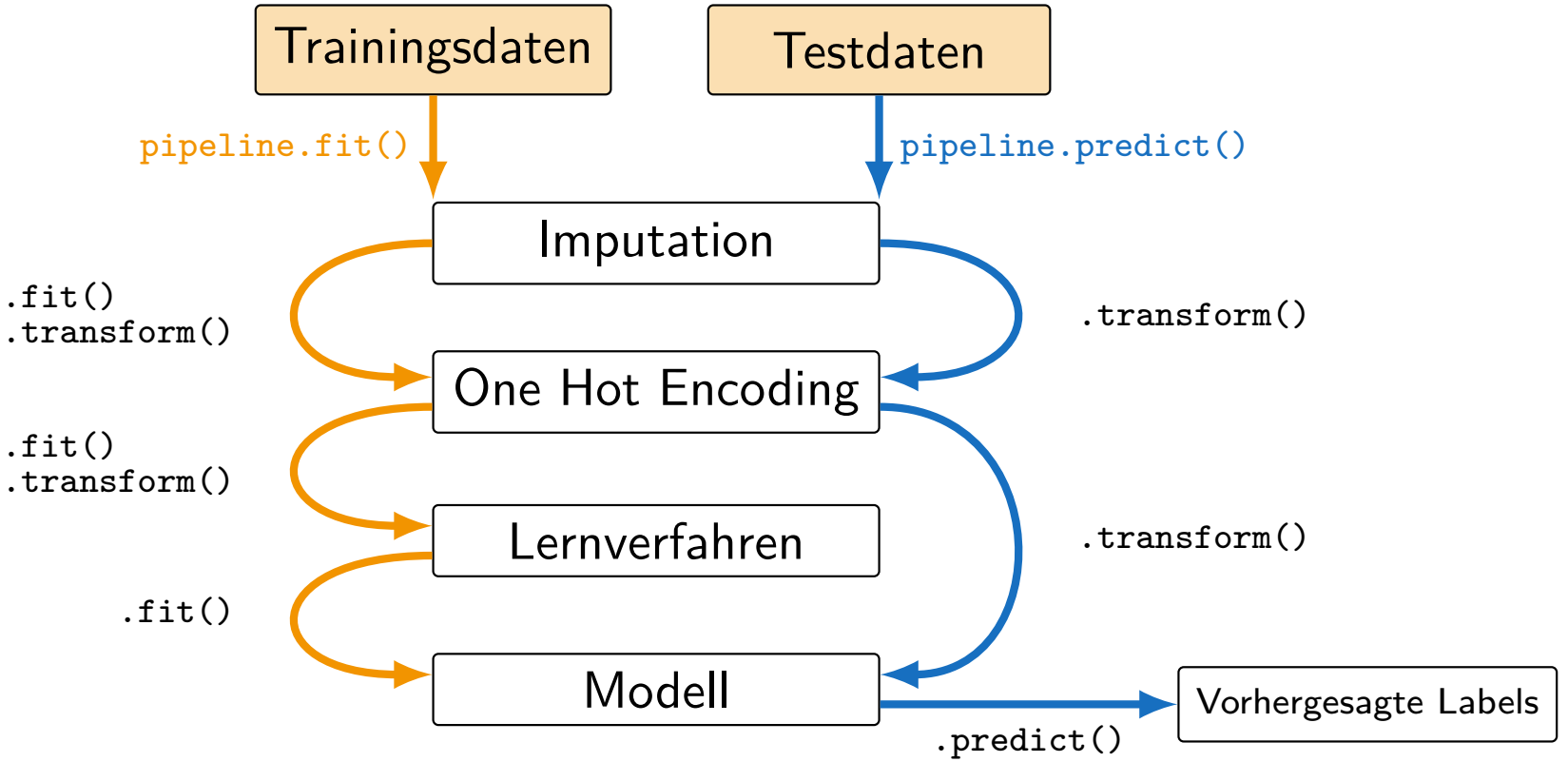
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import OneHotEncoder

df = pd.DataFrame({"price": [12.99, 15.95, 20.31],
 "size": ["M", "XL", "XXL"], "color": ["blue", "red",
  "orange"]})

ct = ColumnTransformer([("ohe",OneHotEncoder(),[2]),
("oe", OrdinalEncoder(), [1])], remainder='passthrough')
ct.fit_transform(df)
# array([[ 1.  ,  0.  ,  0.  ,  0.  , 12.99],
#        [ 0.  ,  0.  ,  1.  ,  1.  , 15.95],
#        [ 0.  ,  1.  ,  0.  ,  2.  , 20.31]])
```

Pipelines in Scikit-learn

- Oft werden beim Machine Learning mehrere Datenvorverarbeitungs- und transformationsschritte hintereinander ausgeführt, bevor ein Klassifikator/Regressor angewendet wird.
- Diese Schritte müssen stets angewendet werden, wenn ein bestehendes Modell auf neue, unbekannte Daten angewendet wird
- In Scikit-learn können diese Schritte in einer Pipeline zusammengefasst werden.
- Eine Pipeline besteht aus einer Sequenz mehrerer Transformatoren, gefolgt von einem Estimator (Modell):



Training und Anwendung einer Pipeline in Scikit-learn



Zusammenfassung

Datenvorverarbeitung ist eine zentrale Aufgabe beim Machine Learning!

- Datentypen, Skalenniveaus
- Typische Datenvorverarbeitungsschritte: Datenzusammenführung, Sampling, Encoding, Diskretisierung, Imputation, Normierung, Standardisierung, Feature Engineering
- Transformers in Scikit-learn
- Pipelines in Scikit-learn

Wie wird es praktisch gemacht?
s. Übung