



Einführung in das Machine Learning

Schulung am 12.05.2025 – 14.05.2025
Ihr Trainer: Nicolas Kuhaft

Überblick Zeiten

7:00 – 8:30



8:45 – 10:15



10:30 – 12:00



13:00 – 14:30

14:30 – 15:00 **OpenSpace**

Inhaltlicher Ablauf

Tag	Thema
1	Einführung Pandas / Numpy
	Einführung in das Machine Learning
	Datenmanagement für Machine Learning
	Erste Prognose für den Machine Learning Task
	Vorstellung der Übungstasks
2	Modellevaluation
	Anwenden unterschiedlicher Modelle
	Ensemble Modellierung
	Anwenden unterschiedlicher Modelle auf Übungstasks
3	Automatisierte Parametersuche
	Regression
	Clusteranalyse
	Neuronale Netze
	Ausblick

Vorstellungsrunde

Einrichtung Pycharm

https://github.com/NKDataConv/machine_learning_schulung_202505

Inhaltlicher Ablauf

Tag	Thema
1	Einführung Pandas / Numpy
	Einführung in das Machine Learning
	Datenmanagement für Machine Learning
	Erste Prognose für den Machine Learning Task
2	Vorstellung der Übungstasks
	Modellevaluation
	Anwenden unterschiedlicher Modelle
	Ensemble Modellierung
3	Anwenden unterschiedlicher Modelle auf Übungstasks
	Automatisierte Parametersuche
	Regression
	Clusteranalyse
	Neuronale Netze
	Ausblick

Code Session Numpy und Pandas

Übung Wetter Daten

/daten/weatherAUS.csv

1. Wo und wann war die höchste Temperatur in Australien?
2. In welcher Stadt ist im Mittel die Temperatur am höchsten?
3. Wenn ich in eine Stadt mit den wenigsten Schwankungen ziehen möchte (gemessen durch Standardabweichung), wo muss ich hinziehen?
4. Welches ist der kälteste Monat? Wann fällt am meisten Regen?
5. Lässt sich eine Erhöhung der Temperatur im Laufe der Jahre feststellen?

Inhaltlicher Ablauf

Tag	Thema
1	Einführung Pandas / Numpy
	Einführung in das Machine Learning
	Datenmanagement für Machine Learning
	Erste Prognose für den Machine Learning Task
2	Vorstellung der Übungstasks
	Modellevaluation
	Anwenden unterschiedlicher Modelle
	Ensemble Modellierung
3	Anwenden unterschiedlicher Modelle auf Übungstasks
	Automatisierte Parametersuche
	Regression
	Clusteranalyse
	Neuronale Netze
	Ausblick

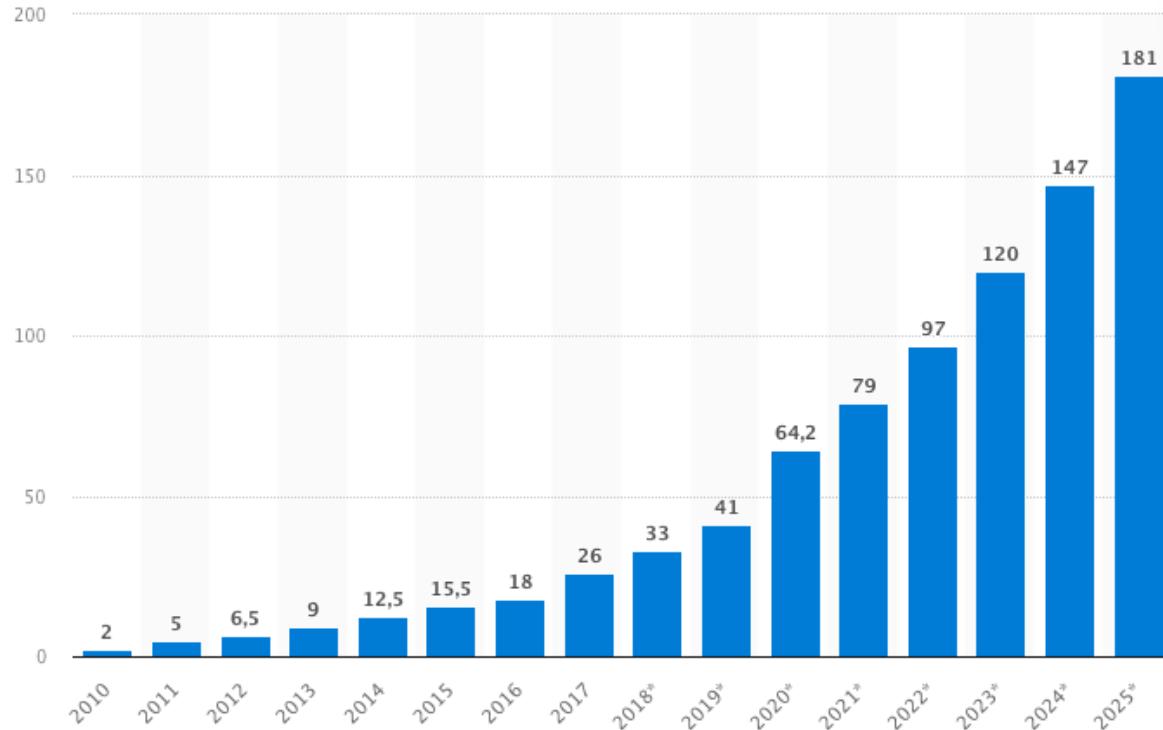


Treibstoff - Daten



Machine Learning Einführung

Datenvolumen in Zettabyte



<https://de.statista.com/statistik/daten/studie/1420298/umfrage/prognose-weltweites-erstelltes-datenvolumen/>



Machine Learning Einführung



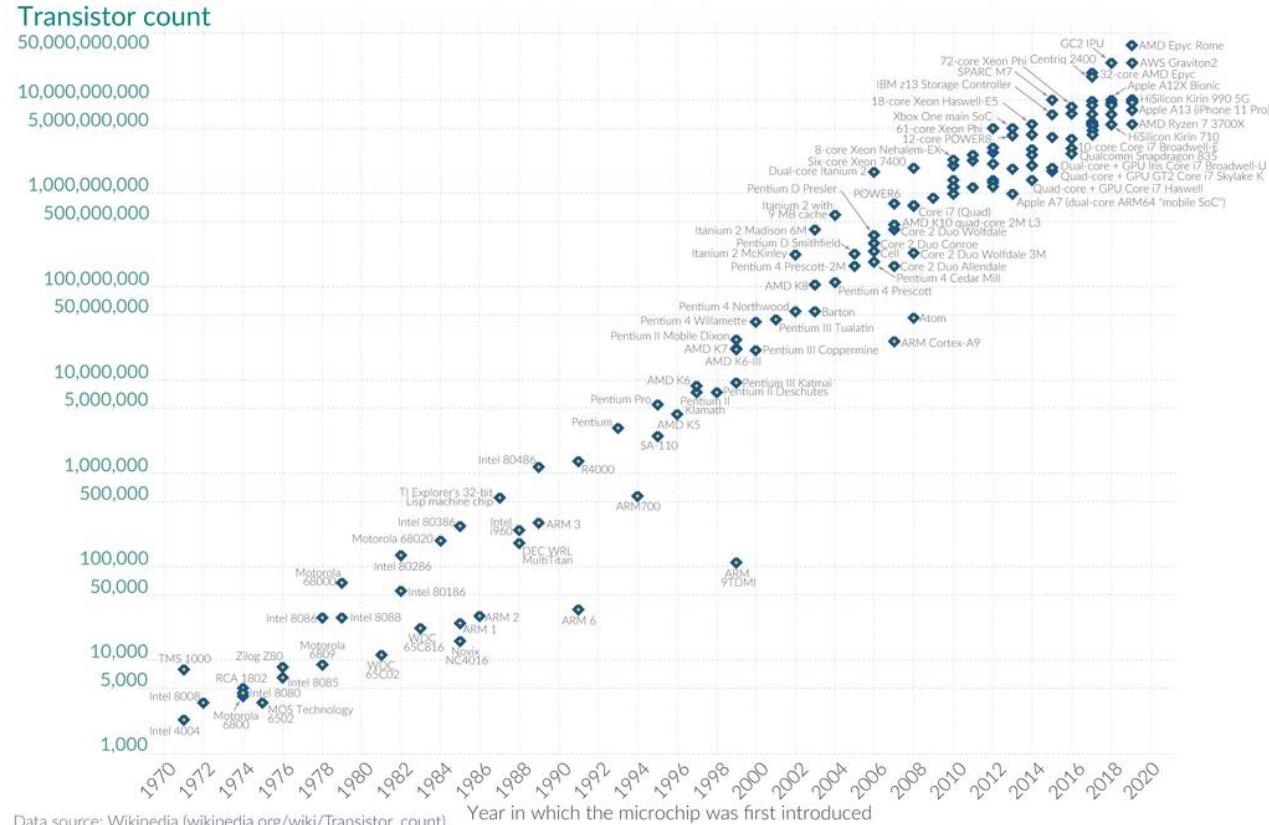
Treibstoff - Daten



Starke Maschinen - GPUs



Machine Learning Einführung



Machine Learning Einführung



Treibstoff - Daten



Starke Maschinen - GPUs

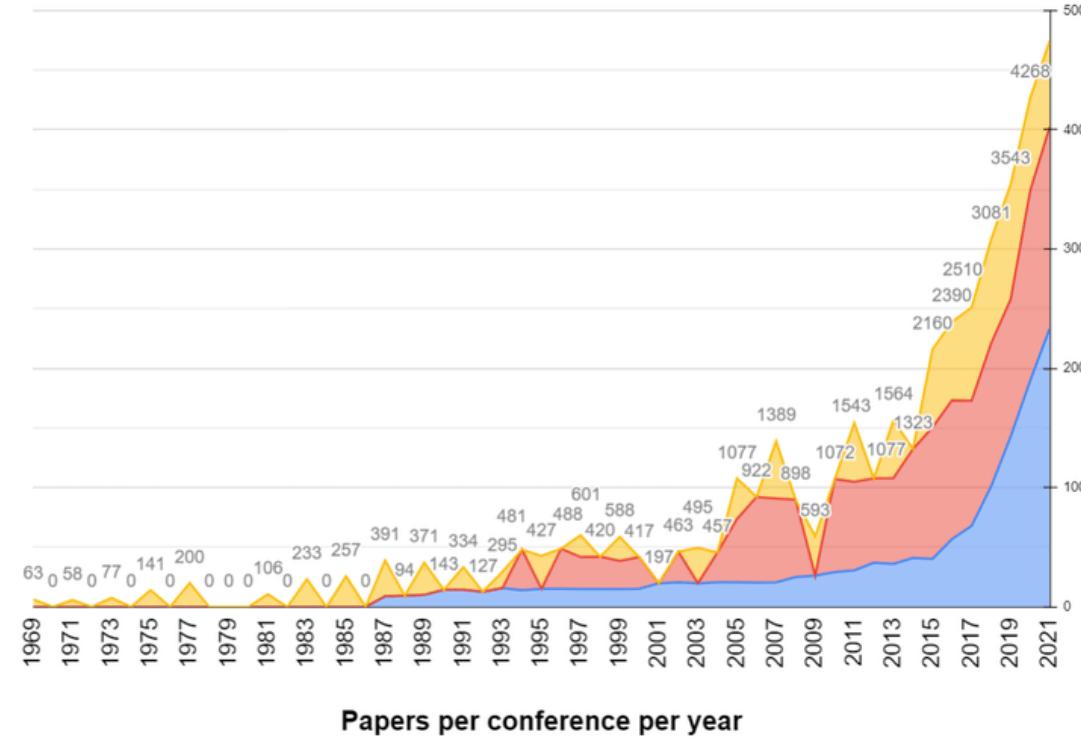


Rocket Science – KI Algorithmen



Machine Learning Einführung

- IJCAI
- AAAI
- NeurIPS



https://www.researchgate.net/figure/Manual-paper-count-per-year-in-AAAI-NeurIPS-and-IJCAI_fig3_360888073



Machine Learning Einführung



Treibstoff - Daten



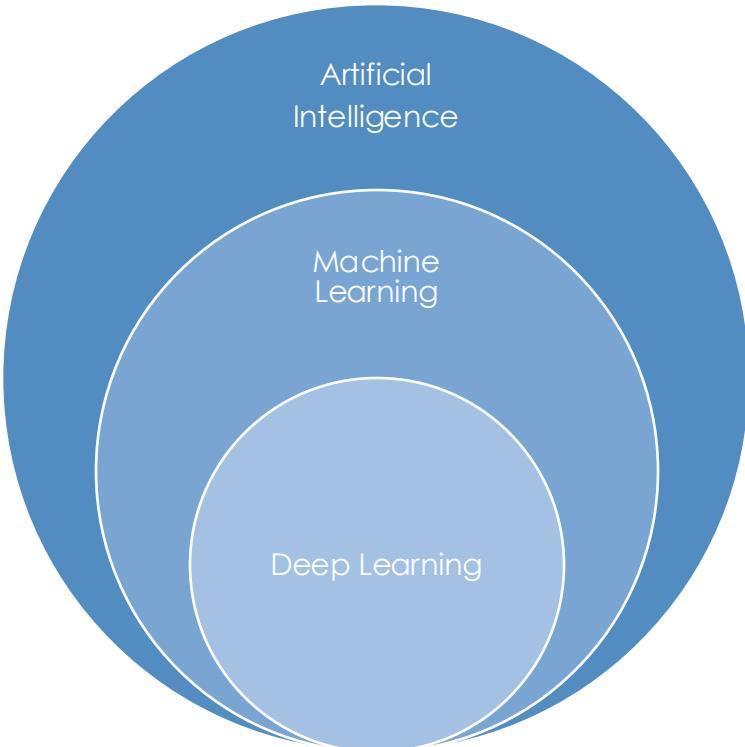
Starke Maschinen - GPUs



Rocket Science – KI Algorithmen



Machine Learning - Einordnung



Artificial Intelligence / Künstliche Intelligenz

Logiken und intelligentes Verhalten von Maschinen.
Keine klare Definition: Was ist intelligent?

Machine Learning

Aus einer Menge von Daten wird eine allg. Regel abgeleitet. Die Ableitung erfolgt automatisiert – d.h. durch einen Algorithmus und ohne explizite Anweisung durch den Programmierer.

Deep Learning

Teilgebiet des Machine Learning. Meint die Anwendung Neuronaler Netze mit vielen verdeckten Schichten. Kann durch seine komplexe innere Struktur komplexe Zusammenhänge in Daten aufzeigen.

Machine Learning ist die Entwicklung von Algorithmen und Modellen, die selbstständig **Muster in Daten** finden – und lernen.

Für welche Art von Problemen eignet sich Machine Learning?

- Wenn das Definieren von Regeln schwer oder nicht möglich ist
 - z.B. Erkennen von Bildern, Verarbeitung von Sprache
 - Im Gegensatz zu Multiplikation von Zahlen

Machine Learning – Eine Einordnung

- Hypothesenfreie Analyse - Modellevaluation durch **Validieren statt statistischer Signifikanz**
- Verfahren mit Techniken aus der Multivariaten Statistik und der Künstlichen Intelligenz [(fast) alle Analysentechniken können zum Machine Learning eingesetzt werden]
- Ziel: Prognose und Mustererkennung
- Einige Techniken haben sich als Kerntechniken des Datamining etabliert:
 - Entscheidungsbäume
 - Random Forest
 - Naive Bayes
 - Neuronale Netzwerke
 - Support Vector Machines

Supervised Learning

= Überwachtes Lernen

Das korrekte Ergebnis ist vorgegeben und der Algorithmus lernt den Zusammenhang.

Klassifikation

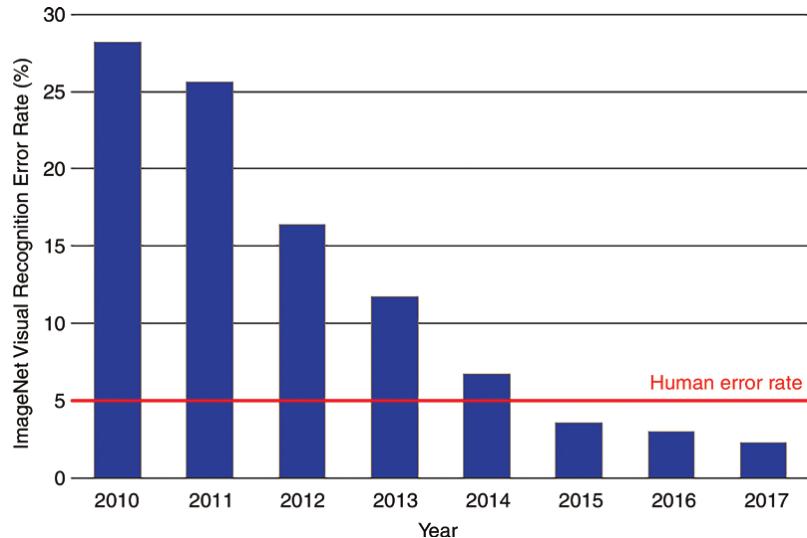
Output ist eine diskrete Klasse, z.B. Vorhersage „Regen-Tag“ oder „kein Regen-Tag“.

Regression

Output ist ein kontinuierlicher Wert, z.B. Temperatur.

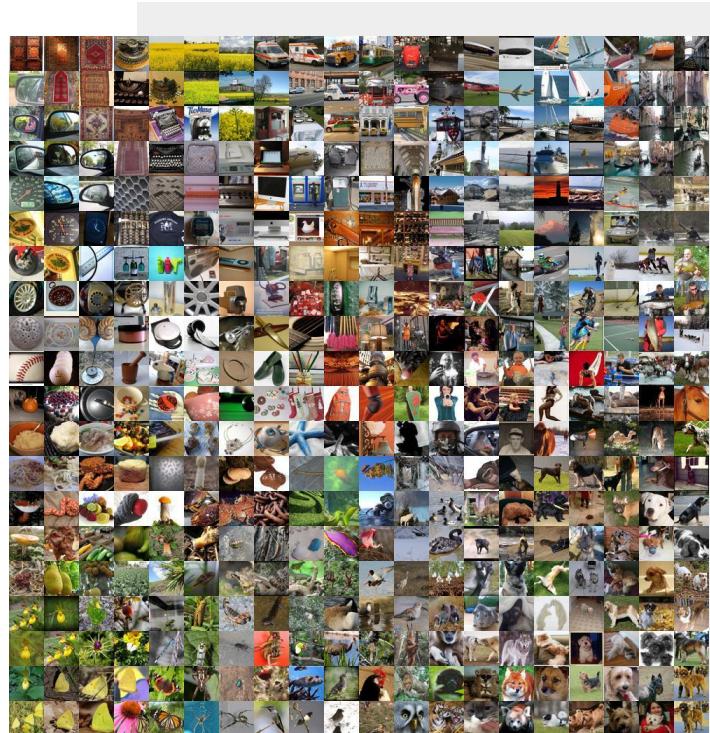
Klassifikation - Imagenet

Einteilung in 1000 Klassen mit über einer Millionen Bildern.



https://www.researchgate.net/figure/Error-rates-on-the-ImageNet-Large-Scale-Visual-Recognition-Challenge-Accuracy_fig1_332452649

Einführung in das Machine Learning

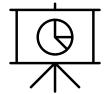


<https://image-net.org/>

- Predictive Maintenance



Input Sensor Maschinen Daten

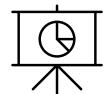


Output: Soll gewartet werden oder nicht (mit Wahrscheinlichkeits-Score)

- Sentiment Analysis



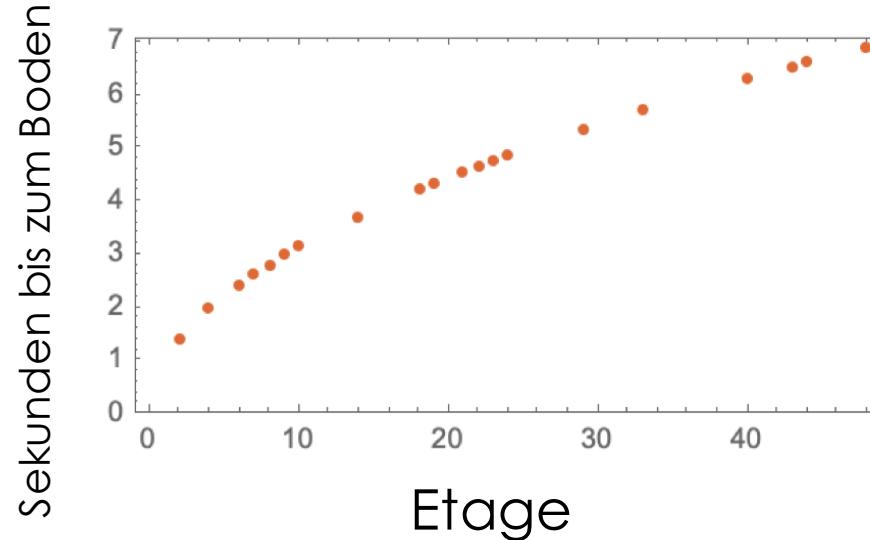
Input: Text, z.B. Kundenbewertung



Output: Klasse positiv, negativ, neutral

Was ist ein Modell?

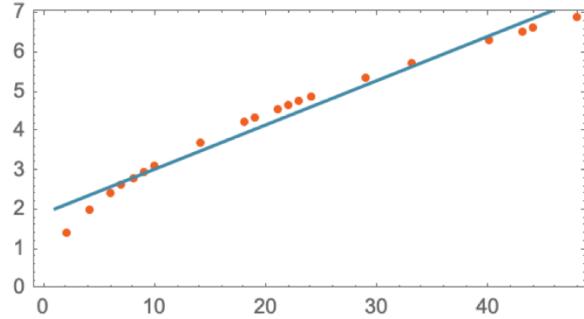
Ein Modell stellt den Zusammenhang zwischen **Features** (Input Variablen) und der Zielgröße her.



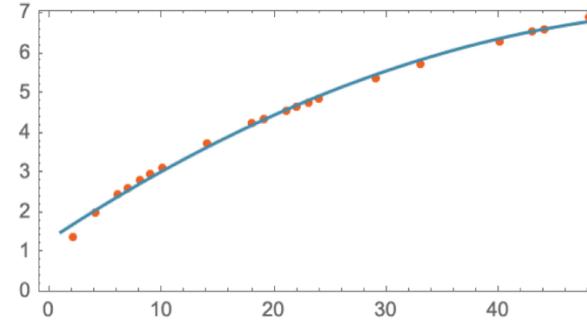
Was ist ein Modell?

a, b und c sind Parameter, die man einstellen kann, damit das Modell die Daten möglichst gut trifft.

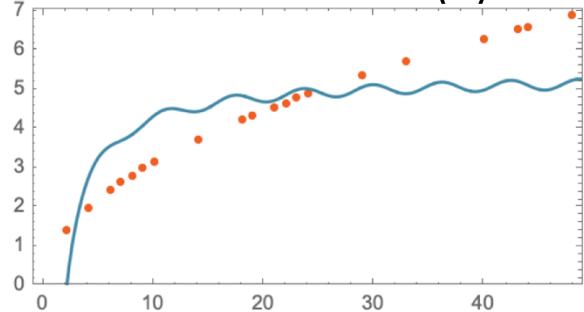
$$a^*x + b$$



$$a^*x + b^*x^2 + c$$



$$a + b/x + c^*\sin(x)$$



Modellevaluation – 2 Herausforderungen

- 1 Eine Herausforderung ist es, den Algorithmus so zu parametrisieren, dass die **Vorhersagegenauigkeit** maximiert wird und gleichzeitig eine **Überanpassung** (Over-fitting) des Modells an die Trainingsdaten zu vermeiden. Es besteht die bei komplexen Modellen die Gefahr, dass der Algorithmus die historischen Daten „auswendig“ lernt und neue Daten nicht gut vorhersagt. Ein gutes Modell ist die **Balance aus maximaler Genauigkeit** der Vorhersage und **abstrakter Sicht** auf die Daten.

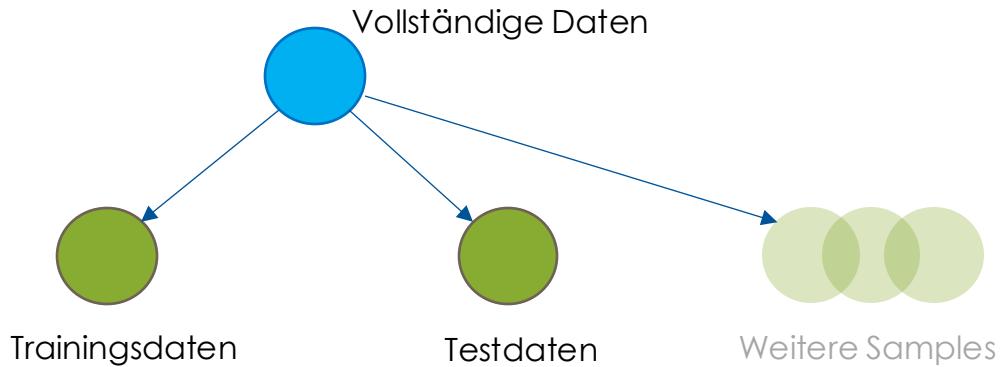


- 2 Eine weitere zentrale Problemstellung des Data Mining ist die **Quantifizierung** der Vorhersagekraft. Die zentrale Fragestellung in diesem Kontext lautet:

„Mit welcher Vorhersagegenauigkeit (Performance) kann ich bei einem erstellten Modell, mit den verwendeten Ausgangsdaten und den spezifisch gesetzten Parametern, rechnen?“

Evaluation der Vorhersagegenauigkeit

Alle Methoden die dazu dienen die **Vorhersagegenauigkeit des Modells zu bestimmen**, haben gemeinsam, dass der Datensatz in 2 oder mehr Teile aufgeteilt wird.

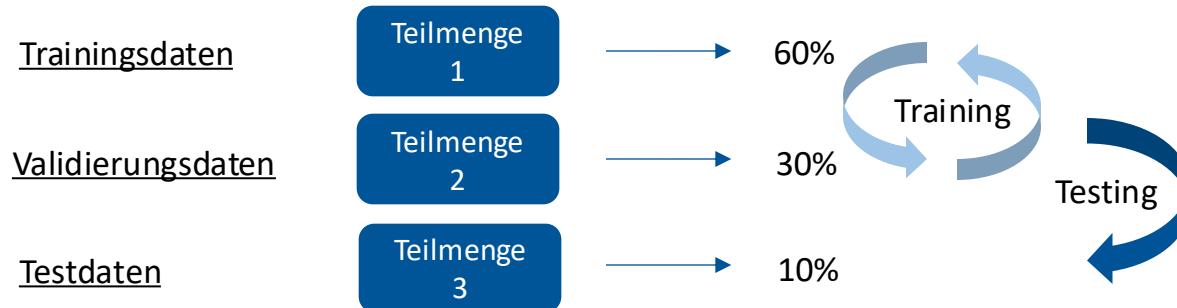


Auf den Trainingsdaten wird das Modell erstellt. Die Testdaten und mögliche weitere Subsets dienen dazu, das Modell ihm unbekannten Daten auszusetzen.

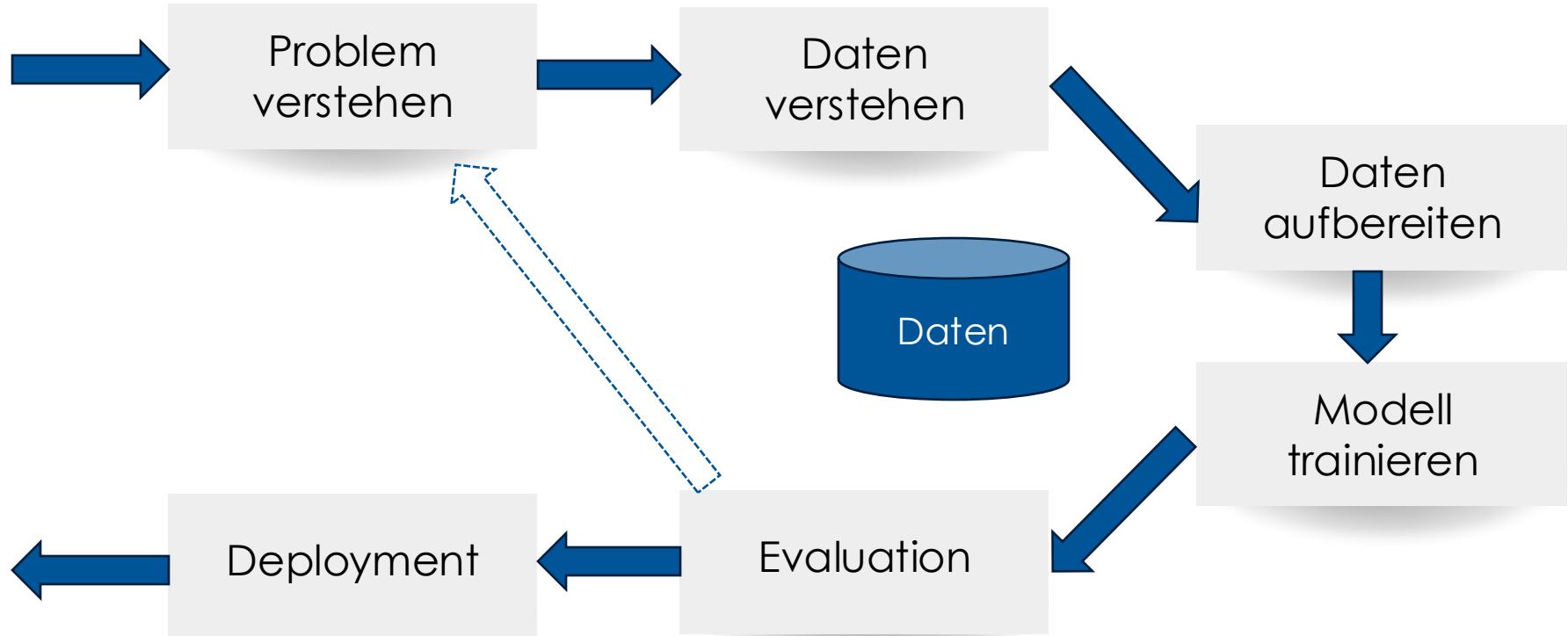
Training (Parametertuning)

Eine Technik die sich diesen Aspekten widmet, ist die **3-Split-Validierung**. Die Datenmenge wird dabei in 3 Teildatensätze geteilt. Auf dem ersten Teildatensatz, den **Trainingsdaten**, wird ein Algorithmus erstellt und dessen Parameter angepasst. Auf der zweiten Teilmenge, den **Validierungsdaten**, wird geprüft, ob der Algorithmus auch auf ihm „fremden“ Daten gute Ergebnisse erzielt (Vermeidung von **Over-Fitting**). Um dabei sicher zu gehen, existiert der dritte Datensatz, der finale **Testdatensatz**.

Die Fehleranteile auf den Validierungs- und Testdaten sollten sich nicht stark voneinander unterscheiden und bilden dann die geschätzte Performance des Modells.



Vorgehen im Machine Learning



Inhaltlicher Ablauf

Tag	Thema
1	Einführung Pandas / Numpy
	Einführung in das Machine Learning
	Datenmanagement für Machine Learning
	Erste Prognose für den Machine Learning Task
	Vorstellung der Übungstasks
2	Modellevaluation
	Anwenden unterschiedlicher Modelle
	Ensemble Modellierung
	Anwenden unterschiedlicher Modelle auf Übungstasks
3	Automatisierte Parametersuche
	Regression
	Clusteranalyse
	Neuronale Netze
	Ausblick

Daten aufbereiten – Feature Engineering

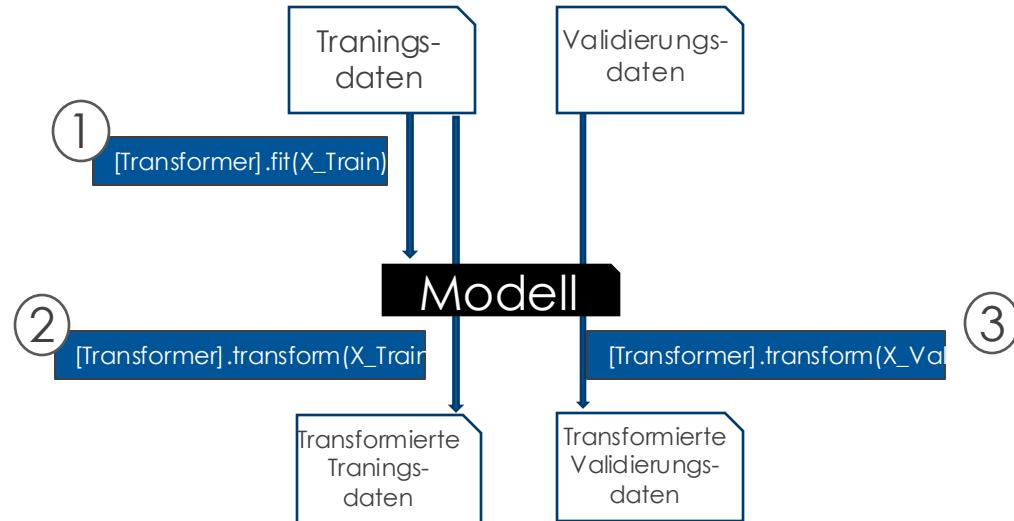
- Typische Probleme:
 - Teilweise Fehlende Werte
 - Werte liegen als Kategorie vor (z.B. m / w / d)
 - Werte müssen skaliert werden
- Mehrwert durch Erstellung neuer Features
 - z.B. aus Datetime Werte das Feature Tag (=0) oder Nacht (=1)
 - Kombination von bestehenden Features (z.B. Aggregationen pro Produkt-Kategorie)

Code Session: Wetter Daten verstehen und aufbereiten

Datenmanagement - Missing Values

Die Datenvorverarbeitung für Machine Learning Anwendungen erfolgt in sklearn über sog. Transformer-Klassen im preprocessing Modul. Transformer-Klassen werden über 2 zentrale Methoden angewendet:

1. **fit** – Mit der Methode fit werden Parameter aus den Trainingsdaten erlernt.
2. **transform** – Nutzt die erlernten Parameter um Daten zu transformieren.



Imputieren fehlender Werte

Mit der Funktion **Imputer** werden **univariate Imputationsmethoden** angewendet. So können fehlende Werte bspw. mit dem arithmetischen Mittel, dem Median oder dem Modus der jeweiligen Verteilung ersetzt werden. In der sklearn Version 0.19 sind diese ungenauen Methoden als einzige implementiert, um fehlende Werte im Datensatz zu ersetzen.

Ab der Version Version 0.20 stehen mit der Funktion **ChainedImputer** elaborativere Techniken zur Missing Value Imputation zur Verfügung. Missing Value Imputationen müssen bis zu diesem Release noch händisch implementiert werden.

```
weather_imputer_float = preprocessing.Imputer(strategy='mean')

weather_imputer_float.fit(weather.loc[:, ['Sunshine', 'WindGustSpeed', 'WindSpeed9am']])

weather.loc[:, ['Sunshine', 'WindGustSpeed', 'WindSpeed9am']] =
    weather_imputer_float.transform(weather.loc[:, ['Sunshine', 'WindGustSpeed', 'WindSpeed9am']])
```

sklearn - Datenrepräsentation

Für die **Modellierung** zur Klassifikation und Regression werden **2 Datenobjekte** benötigt. Zum einen ein **eindimensionales** numpy-Array oder eine pandas-Series und zum anderen eine **zweidimensionale Merkmalsmatrix** (ein Array oder ein DataFrame). Diese beiden Objekte repräsentieren in der statistischen Sprechart die **abhängige bzw. die unabhängigen Variable(n)**. Typischerweise werden diese Variablen mit **y (abhängige Variable)** und **x (unabhängige Variablen)** bezeichnet.

AV
0
1
1
0
1
0
0

UV_1	UV_2	UV_3
2	6	-1
3	7	-2
3	6	0
2	7	0
4	5	-1
2	8	-1
3	8	-4

Label Encoder

Häufig liegen **ordinal oder nominal skalierte Daten als Datentyp "String"** vor. Da viele Methoden und Analysefunktionen in sklearn nicht mit Strings arbeiten können, müssen diese in numerische- (bzw. integer-) Variablen konvertiert werden. Dies kann über eine **eigene Mapper** geschehen oder durch **die Methode LabelEncoder** als sklearn. Insbesondere **dichotome Stringvariablen** können mit dem LabelEncoder umgewandelt und unmittelbar den Algorithmen übergeben werden.

1. Eigene Mapper erstellen:

```
# Erstelle Mapper und inversen Mapper
mapper_WindGustDir = {item: index for index, item in enumerate(pd.unique(weather['WindGustDir']))}
mapper_WindGustDir_inverse = {i[1]:i[0] for i in mapper_WindGustDir.items()}
# Anwenden der Mapper Funktionen
weather[['WindGustDir']] = weather['WindGustDir'].map(mapper_WindGustDir)
weather['WindGustDir'] = weather['WindGustDir'].map(mapper_WindGustDir_inverse)
```

```
data = pd.DataFrame({'Land': ['Hessen', 'NRW', 'Hessen', 'Bayern', 'NRW'],
                     'Jahr': [2008, 2009, 2010, 2011, 2012],
                     'Prozent': [17, 23, 14, 9, 26]})
```

```
LE = preprocessing.LabelEncoder()
LE_fit = LE.fit(data.Land)
LE_fit.transform(data.Land)
```

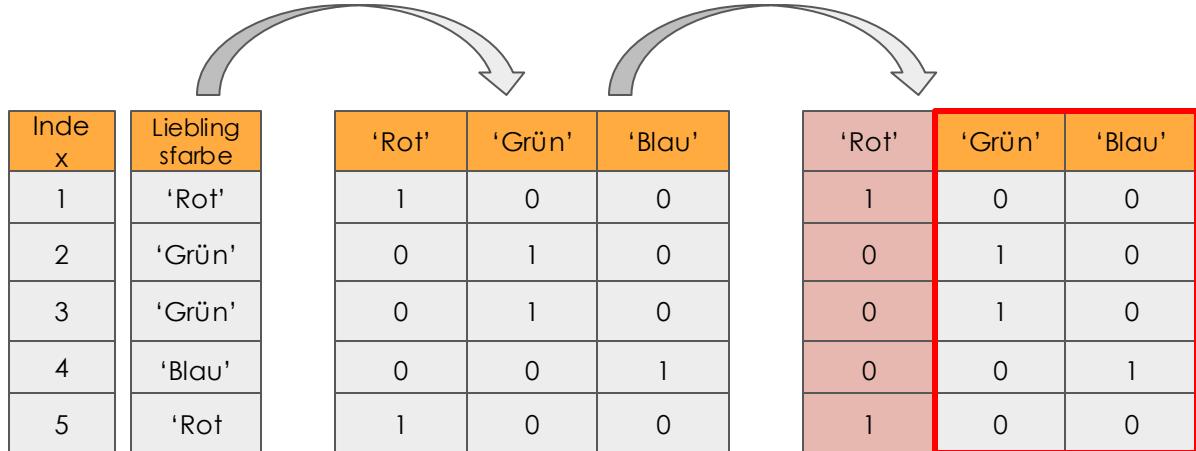
```
Out[49]: array([1, 2, 1, 0, 2], dtype=int64)
```

Datenmanagement mit sklearn - Nominale Daten

Im Vorfeld der Anwendung vieler Analysewerkzeuge in sklearn ist es notwendig, die Daten in die 'richtige' Form zu bringen. Besondere Herausforderungen stellen nominal skalierte Daten dar. Diese müssen zur Eingabe in die Analysealgorithmen dichotomisiert werden. Zwei Methoden stehen dazu zur Verfügung:

1. get_dummies (pandas)
2. OneHotEncoder (aus sklearn)

Dichotomisieren von Daten und auslassen einer Referenzklasse bei Regressionen



Datenmanagement mit sklearn - Nominale Daten

In dem unten stehenden Codebeispiel wird das Vorgehen von `get_dummies()` aus der Pandas Bibliothek demonstriert. Die UrsprungsvARIABLE Land wird gelöscht und Dummyvariablen erzeugt. Unter Auslassung einer Referenzkategorie werden diese nun den Algorithmen übergeben werden.

```
data = pd.DataFrame({'Land': ['Hessen', 'NRW', 'Hessen', 'Bayern', 'NRW'],
                     'Jahr': [2008, 2009, 2010, 2011, 2012],
                     'Prozent': [17, 23, 14, 9, 26]})

pd.get_dummies(data)
```

	Jahr	Prozent	Land_Bayern	Land_Hessen	Land_NRW
0	2008	17	0	1	0
1	2009	23	0	0	1
2	2010	14	0	1	0
3	2011	9	1	0	0
4	2012	26	0	0	1

Datenmanagement mit sklearn - Ordinale Daten

Ordinale Merkmale liegen häufig als Typ String vor und können nicht automatisch in einer korrekten Ordnung gebracht werden. Dies geschieht daher Händisch:

```
df = pd.DataFrame({'Note': ['Ausreichend', 'Mangelhaft', 'Gut', 'Gut', 'Sehr gut', 'Gut', 'Befriedigend']})  
  
mapping = {'Sehr gut': 1,  
           'Gut': 2,  
           'Befriedigend': 3,  
           'Ausreichend': 4,  
           'Mangelhaft': 5,  
           'Ungenügend': 6}  
  
df.Note.map(mapping)  
  
Out[46]:  
0    4  
1    5  
2    2  
3    2  
4    1  
5    2  
6    3  
Name: Note, dtype: int64
```

Datenmanagement mit sklearn – Metrische Daten

Viele Algorithmen generieren bessere Ergebnisse, wenn die Prädiktoren standardisiert sind. Für die Standardisierung von Daten existieren zahlreiche Verfahren, von denen die Wesentlichen in sklearn implementiert sind.

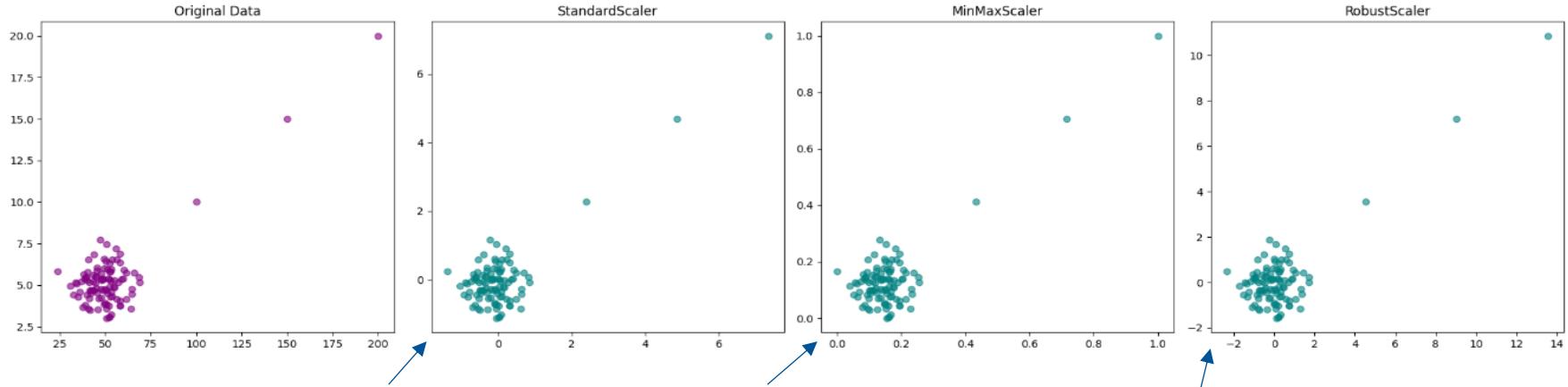
Hier die Anwendung der Klasse MinMaxScaler (Scaler gehören zu den Transformerklassen):

```
from sklearn import preprocessing
x_train_min_max_scaler = preprocessing.MinMaxScaler()
x_train_minmax = x_train_min_max_scaler.fit_transform(x_train)

x_test_min_max_scaler = preprocessing.MinMaxScaler()
x_test_minmax = x_test_min_max_scaler.fit_transform(x_test)
```

Scaler - Wann muss welcher scaler benutzt werden?

Scaler werden benötigt, um Daten in einen einheitlichen Wertebereich zu transformieren, was die Stabilität und Leistung vieler maschineller Lernalgorithmen verbessert.



Skalierte Daten auf Mittelwert 0 und eine Std. 1.

Einsatzgebiet: Bei Modellen, die auf Distanzen zwischen Datenpunkten basieren (z. B. lineare Modelle, k-NN, PCA).

Vorteile: Stabilisiert die Varianz der Merkmale und weniger anfällig für Ausreißer.

Nachteile: Extreme Ausreißer können verzerrn.

Transformiert auf bestimmten Bereich (bspw. 0 und 1)

Einsatzgebiet: Bei Methoden, die nicht auf Verteilungen sondern auf Gradienten basieren (z. B. neuronale Netze).

Vorteile: Erhält das Verhältnis der Datenpunkte und vermeidet negative Werte.

Nachteile: Empfindlich gegenüber Ausreißern.

Ähnlich wie StandardScaler, jedoch nutzt er Median und Interquartilsabstand verwendet --> Ausreißereffekte werden minimiert.

Einsatzgebiet: Sinnvoll bei Daten mit vielen Ausreißern.

Vorteile: Weniger anfällig für Ausreißer.

Nachteile: Verändert die Form der Verteilung. Weniger präzise, wenn die Daten keine starken Ausreißer aufweisen.

Aufteilen der Daten

Das **Aufteilen der historischen Daten** in Trainings- und Testdaten kann mit der Utility-Funktion `train_test_split` aus dem **Modul model_selection** durchgeführt werden. Die Funktion nimmt eine beliebige Anzahl von Argumenten entgegen und teilt alle Objekte mit identischer Fallauswahl in 2 Teildatensätze.

In dem unten aufgeführten Beispielaufruf werden die Objekte `weather` und `target` übergeben. Entsprechend werden 4 Objekte durch die Funktion returniert.

```
from sklearn import model_selection
x_train, x_vali, y_train, y_vali = model_selection.train_test_split(weather,
                                                               target,
                                                               test_size = 0.3,
                                                               random_state = 5)
```

Inhaltlicher Ablauf

Tag	Thema
1	Einführung Pandas / Numpy
	Einführung in das Machine Learning
	Datenmanagement für Machine Learning
	Erste Prognose für den Machine Learning Task
	Vorstellung der Übungstasks
2	Modellevaluation
	Anwenden unterschiedlicher Modelle
	Ensemble Modellierung
	Anwenden unterschiedlicher Modelle auf Übungstasks
3	Automatisierte Parametersuche
	Regression
	Clusteranalyse
	Neuronale Netze
	Ausblick

Descision- und Regression Trees



Vorteile:

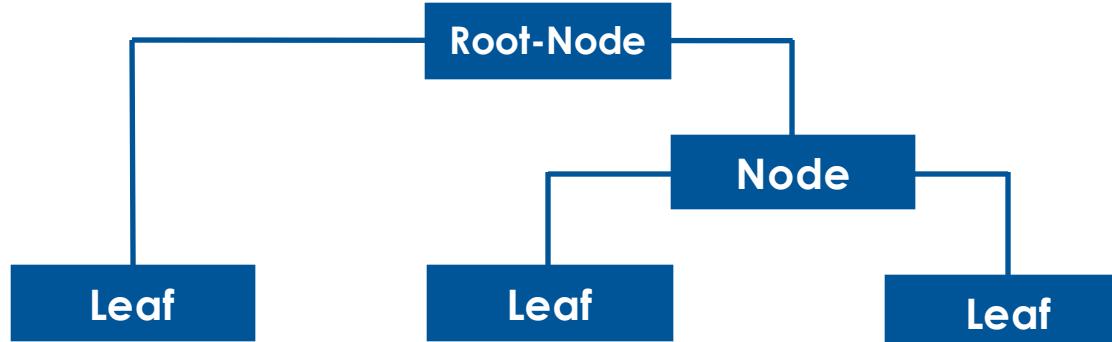
- Nachvollziehbarkeit (White-Box)
- Visualisierbar
- Übersicht über Feature-Importance
- Geringes Datenmanagement: Normalisierung nicht notwendig

Nachteile:

- Kurzsichtigkeit: Der Algorithmus entscheidet iterativ, welcher Split den größten Informationsgewinn erzielt.
- Interaktionseffekte zwischen den Features können durch Baumverfahren nicht aufgedeckt werden.
- Die Gesamtvariabilität wird nicht ausgenutzt (nicht alle Variablen werden einbezogen)
- Instabil: Geringe Änderungen in den Daten können den Baum sehr verändern.
- Bias für unbalancierte Klassengrößen
- Das Model ist nicht extrapolierbar (was allerdings auch ein Vorteil sein kann, da so keine Extremwerte in der Vorhersage entstehen)

Decision- und Regression Trees - Terminologie

Entscheidungs- und Regressionsbäume bestehen aus einer **Wurzel**, einer oder mehreren **Knoten** und aus **Blättern**. Ein Blatt ist ein Knoten, der nicht weiter verzweigt wird. Je nach Verfahren und Datenlage teilen sich die Knoten in zwei oder mehrere Verzweigungen auf.

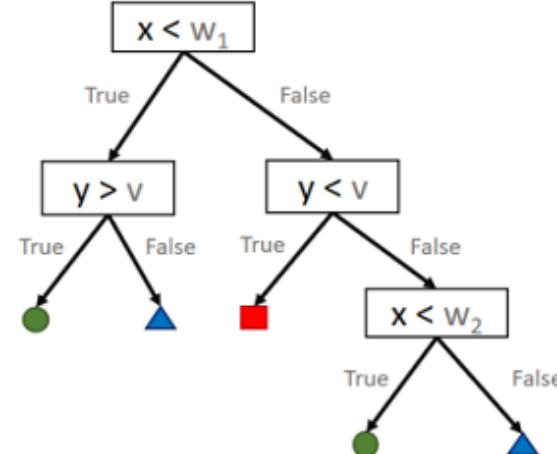
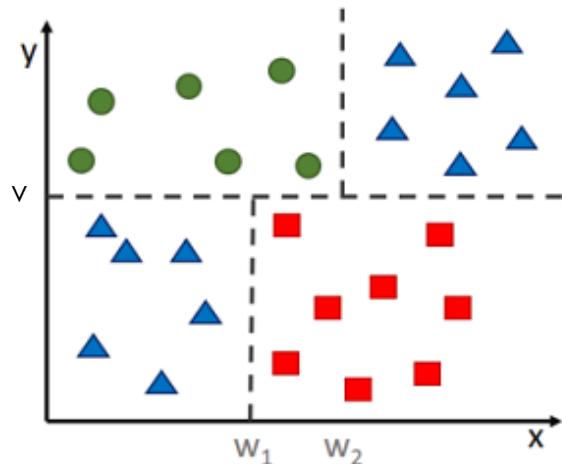


Decision- und Regression Trees

Je nachdem, ob die Zielvariable kategorial oder metrisch ist spricht man von Entscheidungsbäumen (kategoriale Zielvariable) oder Regressionsbäumen (metrische Zielvariable).

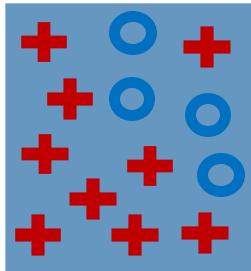
Auf der ersten Ebenen des Baums identifiziert der Algorithmus diejenige Variable, die die **größte Diskriminanz** hinsichtlich der Zielvariable liefert. Der Datensatz wird somit in zwei Teile geteilt. Für die **jeweiligen Teildatensätze** wird wiederum diejenige Variable gesucht, die für diesen Teildatensatz die höchste Diskriminanz liefert usw.

Dieses Vorgehen wird als **rekursive Partitionierung (recursive partitioning)** bezeichnet und ist das Kernelement der Entscheidungs- und Regressionsbäume.



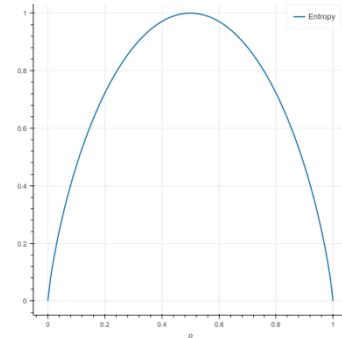
Decisiontrees – Entscheidungsregeln mit Entropie

Initiale Verteilung

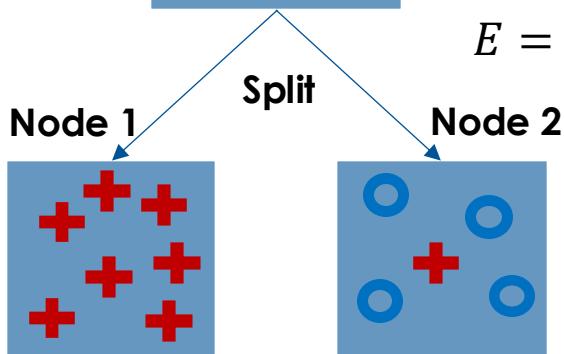


Das vorrangige Ziel des Entscheidungsbaumes ist es, die Unordnung in den Daten zu beseitigen. Typischerweise wird das Maß an Unordnung als sog "Entropie" ausgedrückt. Mit jedem Split soll die Entropie gesenkt werden.

$$p(+)=9 / 13 = 0,7$$
$$p(\circ)=4 / 13 = 0,3$$

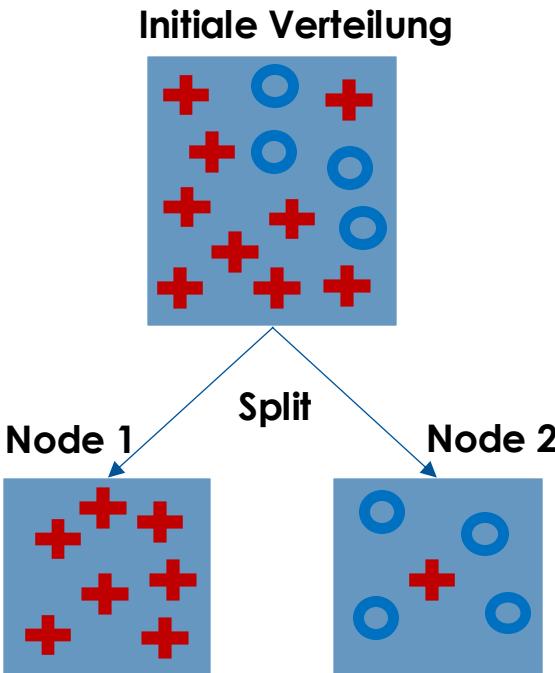


$$E = - p(+)*\log_2(p(+)) - p(\circ)*\log_2(p(\circ))$$



```
import math
-0.4 * math.log(0.4,2) - 0.6 * math.log(0.6,2) = 0.97
-0.5 * math.log(0.5,2) - 0.5 * math.log(0.5,2) = 1
-0.6 * math.log(0.6,2) - 0.4 * math.log(0.4,2) = 0.97
-0.7 * math.log(0.7,2) - 0.3 * math.log(0.3,2) = 0.88
-0.8 * math.log(0.8,2) - 0.2 * math.log(0.2,2) = 0.72
-0.9 * math.log(0.9,2) - 0.1 * math.log(0.1,2) = 0.47
-0.99 * math.log(0.99,2) - 0.01 * math.log(0.01,2) = 0.08
```

Decisiontrees – Entscheidungsregeln mit Gini



Neben der Entropie wird häufig auch der sog. Gini-Index (auch Gini-Impurity) verwendet, um das Maß an "Unordnung" in den Daten anzugeben. Ganz allg. berechnet er sich wie folgt:

$$G_{(\text{mod})} = \sum_{i=1}^n p(i) * 1 - p(i)$$

Für das nebenstehende Beispiel ergeben sich daraus Werte von:

Initiale Verteilung:

$$G_{(\text{init})}: 0,7 * (1-0,7) + 0,3 * (1-0,3) = 0,42$$

Node 1:

$$G_{(\text{node1})}: 1 * (1-1) + 0 * (1-0) = 0$$

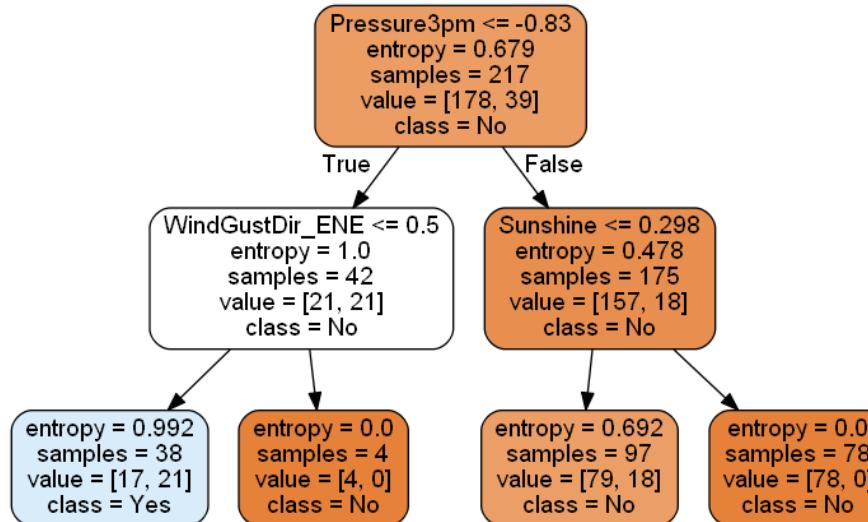
Node 2:

$$G_{(\text{node2})}: 0,2 * (1-0,2) + 0,8 * (1-0,8) = 0,32$$

Decision- und Regression Trees – Visualisierung / Interpretation

Die Regeln stellen den Entscheidungsbaum in Form einer logischen Aussage.

WENN der Luftdruck um 15h $> -0,83$ **UND** die Sonnenscheindauer $\leq 0,298$ Stunden, **DANN** beträgt die Wahrscheinlichkeit, dass es am folgenden Tag nicht regnet, (79/97) 81%.



Code Session

Übung

1. Variiere die Modellparameter. Mit welchen Parametern erhältst du die beste Performance?
2. Feature Engineering: Erstelle das Feature „Monat“ und erstelle eine sinnvolle Transformation. Verbessert sich die Performance? Wie wirkt sich das zusätzliche Feature auf das Overfitting aus?
3. Verwende zusätzlich das Feature „City“. Welche Transformation ist sinnvoll? Wie wirkt sich die Einbeziehung auf das Ergebnis aus? Wie wirkt sich das zusätzliche Feature auf das Overfitting aus?
4. Verbessert sich die Performance durch das Weglassen von Features?
5. Welche weiteren Features könnten sinnvoll sein? Verbessern sie das Ergebnis?

Decision- und Regression Trees – Klasse "DecisionTreeClassifier"

```
class sklearn.tree
```

```
DecisionTreeClassifier(criterion='gini',  
                      splitter='best',  
                      max_depth=None,  
                      min_samples_split=2,  
                      min_samples_leaf=1,  
                      min_weight_fraction_leaf=0.0,  
                      max_features=None,  
                      random_state=None,  
                      max_leaf_nodes=None,  
                      min_impurity_decrease=0.0,  
                      min_impurity_split=None,  
                      class_weight=None,  
                      presort=False)
```

criterion: "gini" oder "entropy"

Die Performance ändert sich durch für beide Ausprägungen nicht. Gini ist schneller zu berechnen da entropy Logarithmen nutzt.

splitter: "best" oder "random"

"best" sorgt dafür, dass ein Split am geeigneten Feature gemacht wird. "random" wählt ein

Feature zufällig aus. "random": Weniger nützlich; vermeidet Overfitting; nutzt auch unwichtige/
irrelevante Features; in Ensembles kann dieses Argument hilfreich sein.

max_depth: Maximale Tiefe des Baumes (Anzahl Ebenen von Root-Node zu Leaf)

min_samples_split: Min. Anzahl von Fällen für einen weiteren Split.

min_samples_leaf: Min. Anzahl von Fällen in einem Leaf.

min_weight_fraction_leaf: Min. Summe aller Fallgewichtungen in einem Leaf.

max_features: Anzahl der Features die für einen weiteren Split betrachtet werden.

max_leaf_nodes: Max. Anzahl der Leaf-Nodes.

min_impurity_decrease: Minimale Verbesserung des Modells nach Aufteilung

min_impurity_split: Deprecated (jetzt "min_impurity_split")

class_weight: Dictionary mit Klassengewichten der Form: {class_label: weight}

sklearn – Schätzer API

Sklearn ist die umfassendste Sammlung von gebräuchlichen Algorithmen im Kontext von Machine Learning in Python.
Vier Aspekte machen sklearn so wertvoll:

1. **Vielfältigkeit** der implementierten **Algorithmen** (*Regression- und Klassifikationsverfahren, Dimensionsreduktion, Clustering etc.*)
2. **Einheitliche** Verwendung von **Algorithmen** (*Klare Schnittstellendefinition*)
3. Umfassende **Onlinedokumentation**
4. Unterstützt **Algorithmenverkettungen** (*Machine Learning Anwendungen sind eine Verkettung von Datenmanagement- und Analysealgorithmen – in sklearn ist diese Verkettung programmatisch umgesetzt*)

sklearn – Verwendung der Schätzer API

Die Verwendung der Schätzer API erfordert in der Regel folgende Vorgehensschritte:

1. **Datenmanagement** (Aufteilen in Trainings- und Testdaten, erstellen des Ziel- und Merkmalsobjektes)

```
y = dat['Endergebnis']
x = dat.loc[:, ['MA_Heim', 'MA_Ausw', 'MA_Heim_Tore', 'MA_Ausw_Tore', 'Pos_Heim', 'Pos_Ausw']]
```

2. **Importieren** der Schätzerklasse

```
from sklearn import tree
```

3. **Instanzieren** der Klasse mit den entsprechenden Hyperparametern

```
cls = tree.DecisionTreeClassifier(min_samples_split = 3,
                                   max_depth=7)
```

4. **Anpassen** des Modells an die Daten mit der **fit-Methode**

```
cls.fit(X=x_train, y=y_train)
```

5. **Anwenden** des Modells auf neue Daten mit der **predict-Methode**

```
preds = cls.predict(X=x_test)
preds_proba = cls.predict_proba(x_test)
```

Trainieren der Algorithmen

Die Implementierung und Anwendung der unterschiedlichen klassischen Data Mining Algorithmen ist sehr homogen und erfolgt in 2 Schritten.

1. Es wird ein Modellobjekt mit den gewünschten Parametern erzeugt.
2. Das Modellobjekt verfügt über eine Methode `fit()`, dem die Trainingsdaten übergeben werden.

```
from sklearn import tree

tree_clf = tree.DecisionTreeClassifier(max_depth=3)
tree_clf.fit(x_train, y_train)
```

Vorhersage ‚neuer‘ Werte

Die Vorhersage neuer Werte erfolgt über eine Methode ‚predict‘ des Modellobjekts.

Bei Regressionsproblemen (und damit einer metrischen Zielvariablen) entspricht der Rückgabewert der predict-Funktion der eigentlichen Vorhersage.

Bei Klassifikationsproblemen kann eine Prognose für die Klassenzugehörigkeit ausgeben werden (predict-Funktion) oder die Wahrscheinlichkeit für eine bestimmte Klassenzugehörigkeit (predict_proba). Anschließend kann die Klassenzugehörigkeit unter Verwendung eines user-definierten Cutoff-Wertes selbst zugewiesen werden.

```
# --- Default Cutoff bei 0.5
tree_predictions = tree_clf.predict(x_vali)

# --- Nutzer definierter Cutoff-Wert
tree_predictions_proba = tree_clf.predict_proba(x_vali)
tree_predictions_cutoff = [1 if i > 0.3 else 0 for i in tree_predictions_proba[:,1]]
```

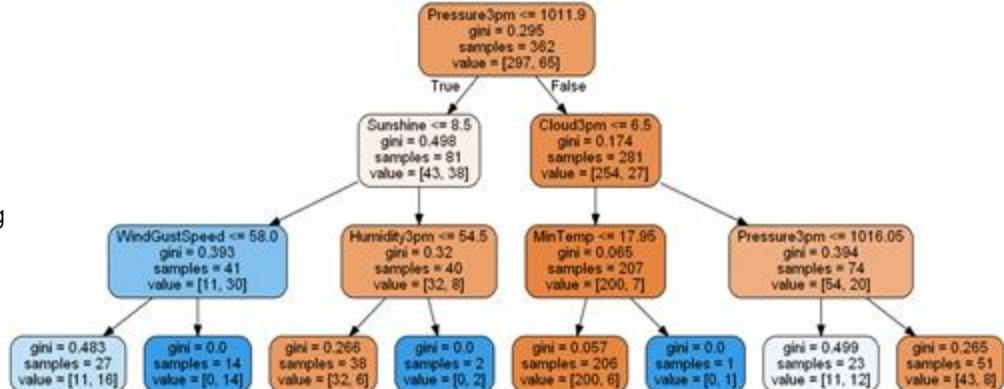
Visualisieren von Decision-Trees

In sklearn erstellte Entscheidungsbäume können durch die externe Software graphviz visualisiert werden. Graphviz ist eine Open-Source Software und unter der URL: <http://graphviz.org> erhältlich. Nach der Installation ist es notwendig den Pfad zum bin-Ordner in die Umgebungsvariable path aufzunehmen. Mit der Bibliothek pydotplus können Sie anschließend aus der Python-Umgebung die Grafiken exportieren. pydotplus lässt sich einfach mit pip installieren.

Schritte zur Visualisierung von Entscheidungsbäumen:

1. graphviz installieren
2. Umgebungsvariable anpassen
3. pydotplus mit pip installieren
4. Evtl. die Dependencies graphviz und pyparsing mit pip manuell installieren

```
from pydotplus import graph_from_dot_data
from sklearn.tree import export_graphviz
...
tree_clf.fit(X_data, Y_data)
dot_data = export_graphviz(tree_clf, filled = True, rounded = True, class_names=['No', 'Yes'],
feature_names=list(X_data.columns), out_file=None)
graph = graph_from_dot_data(dot_data)
graph.write_png("tree.png")
```



Inhaltlicher Ablauf

Tag	Thema
1	Einführung Pandas / Numpy
	Einführung in das Machine Learning
	Datenmanagement für Machine Learning
	Erste Prognose für den Machine Learning Task
Vorstellung der Übungstasks	
2	Modellevaluation
	Anwenden unterschiedlicher Modelle
	Ensemble Modellierung
	Anwenden unterschiedlicher Modelle auf Übungstasks
3	Automatisierte Parametersuche
	Regression
	Clusteranalyse
	Neuronale Netze
Ausblick	



Code Session Übungstask

Übung

1. Variiere die Parameter. Wieviel Gewinn / Verlust erhältst du? Welches sind die wichtigsten Features? Wie hoch ist das Overfitting?

Inhaltlicher Ablauf

Tag	Thema
1	Einführung Pandas / Numpy
	Einführung in das Machine Learning
	Datenmanagement für Machine Learning
	Erste Prognose für den Machine Learning Task
	Vorstellung der Übungstasks
2	Modellevaluation
	Anwenden unterschiedlicher Modelle
	Ensemble Modellierung
	Anwenden unterschiedlicher Modelle auf Übungstasks
3	Automatisierte Parametersuche
	Regression
	Clusteranalyse
	Neuronale Netze
	Ausblick

Klassifikation - Unbalancierte Modelle

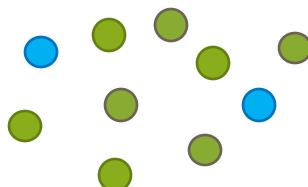
Bei Klassifikationsproblemen kann eine **unausgewogene Verteilung der Zielvariable** zu Problemen bei der Beurteilung der Modellperformance kommen. Angenommen werden soll **folgende Datenlage**:

1000 Testdaten inklusive der Information **HIV-Infektion Ja/Nein**. Mit **Data Mining** Methoden soll nun anhand der Testdaten die **Infektion klassifiziert** werden.

Angenommen sei nun, dass in dem Datensatz lediglich **20 Infizierte Personen** enthalten sind. **Angenommen** sei darüber hinaus, dass das Modell zwar **nicht infizierte Personen sehr gut bestimmen** kann, **tatsächlich infizierte** jedoch **tendenziell auch als nicht infiziert** ausweist.

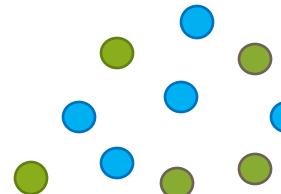
Über **alle Fälle** würde zwar eine **sehr gute Modellperformance** in einer Grundgesamtheit bestehen, die ähnlich wie die Zielvariable bei der Modellbildung verteilt ist. Angewendet auf eine **andere Gesamtheit** (bspw. mit ausgewogener Verteilung wie in einer Klinik) würde die **Performance** jedoch **wesentlich schlechter** ausfallen.

Datenlage bei der Modellierung



Gute Performance durch hohen Anteil der Klasse, die gut vorhergesagt werden kann.

Datenlage bei der Modelanwendung



Schlechte Performance durch hohen Anteil der Klasse, die nicht gut vorhergesagt werden kann.

Aus dem Beispiel der vorangegangenen Folie zeigt sich, dass die **Minimierung der Gesamtfehlers nicht** immer das **alleinige Gütekriterium** eines Modells ist. 2 Methoden dienen dem Umgang mit unbalancierten Modellen:

1. **Stratifizierung** des Trainingssets

Aus dem Originaldatenset wird eine Gleichverteilung gesampelt. Unterschieden wird hier Undersampling und Oversampling.

Undersampling: Aus der Majoritätsklasse werden Fälle eliminiert, bis Gleichverteilung entsteht.

Oversampling: Aus der Minoritätsklasse werden Fälle dupliziert, bis Gleichverteilung entsteht.

2. Nutzen **weiterer Metriken** zur Modellbeurteilung

Klassifikation - Modellevaluation

Die einfachste Form zur Bestimmung der Modelperformance bei Klassifikationsproblemen ist der relative Fehler der Klassifikation.

Dieser hat einen Wertebereich von 0 bis 1 und ermittelt sich durch die Anweisung: $= (\text{Falsche_Klassifikationen} / \text{Fallzahl})$

Es gilt: Je kleiner der relative Fehler des Modells, desto besser die Vorhersagegenauigkeit.

		Tatsächlich	
		Positiv	Negativ
Prognose	Positiv	Echt positiv 10	Falsch Positiv 5
	Negativ	Falsch Negativ 12	Echt Negativ 110

$$= (12+5) / (10+5+12+110) = 12\%$$

Klassifikation - Weitere Metriken der Modellevaluation

- **Präzision**

Der Anteil der **echt Positiv**-Werte an den positiv **prognostizierten** Werten insgesamt: $\frac{A}{A+B}$

- **Sensitivität (auch Recall)**

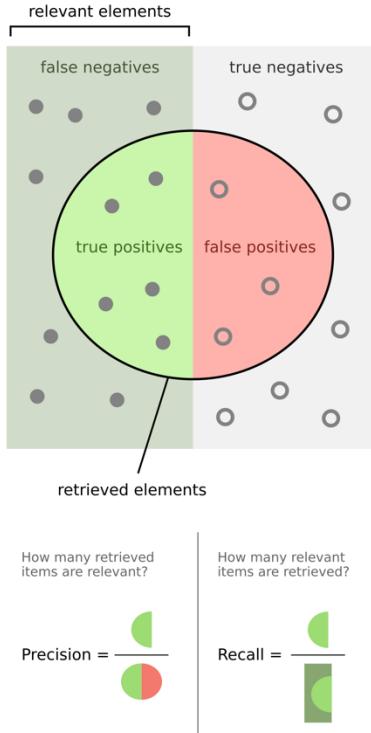
Der Anteil der **echt Positiv**-Werte an den **tatsächlich** positiven Werten: $\frac{A}{A+C}$

- **Spezifität**

Der Anteil der **echt Negativ**-Werte an den **tatsächlich** negativen Werten: $\frac{D}{B+D}$

		Tatsächlich	
		Positiv	Negativ
Prognose	Positiv	Echt positiv A	Falsch Positiv B
	Negativ	Falsch Negativ C	Echt Negativ D

Accuracy, F1-Score, Precision und Recall



- **Accuracy** misst den Anteil korrekt klassifizierter Beispiele, sinnvoll bei **ausbalancierten** Datensätzen.
- **Precision** definiert den Anteil der als positiv vorhergesagten Beispiele, die tatsächlich positiv sind.
- **Recall** misst, wie viele der tatsächlich positiven Beispiele korrekt erkannt werden.
- Der **F1-Score** ist das harmonische Mittel aus Precision und Recall und hilft bei unausgewogenen Daten.
- Diese Metriken sind besonders relevant, wenn Klassen stark **unterschiedlich häufig** vorkommen.

<https://upload.wikimedia.org/wikipedia/commons/thumb/2/26/Precisionrecall.svg/1200px-Precisionrecall.svg.png>

Code Session – Precision und Recall berechnen

Übung – Evaluierung Wetter Daten

1. Ändere den Cutoff Wert, um den Recall zu verbessern.
2. Welcher Cutoff Wert muss gewählt werden, um mindestens 80% der Regentage korrekt vorherzusagen?

Klassifikation - Kostenmodellierung

Durch das Beispiel wird deutlich, dass **richtige und falsche Klassifikationen** mit **unterschiedlichen Kosten** versehen sein können. Einen nicht-HIV-infizierten als infiziert einzustufen, scheint weniger dramatisch als einen HIV-Infizierten als nicht-infiziert zu klassifizieren. Bei der **Kostenmodellierung** werden die **Fehlklassifikationen** mit einem **numerischen Faktor** versehen.

Hier wird eine Falsch-Negativ Prognose mit **100x höheren Kosten** versehen als die Falsch-Positiv-Prognose.

		Tatsächlich	
		Positiv	Negativ
Prognose	Positiv	Echt positiv 0	Falsch Positiv 1
	Negativ	Falsch Negativ 100	Echt Negativ 0

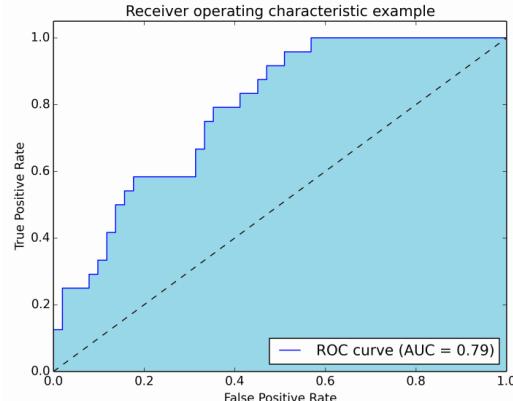
Klassifikation - Cutoff-Auswahl

Um das Modell hinsichtlich der **Kostenmatrix** zu **optimieren**, kommt die sog. **ROC-Kurve** zum Einsatz. Anhand dieser lässt sich erkennen, wie die Fehlerverteilung bei unterschiedlicher Modellkonfiguration ausfällt. Modellkonfiguration meint: Die Verteilung der Fehlerklassen mit unterschiedlichem Cutoff-Wert wird abgebildet.

```
predictions_proba = clf.predict_proba(x_vali)

predictions_cutoff = [1 if i > 0.3 else 0 for i in predictions_proba[:,1]]
```

	Actual	Probability	Class_Prediction
0	1	0.93750	1
1	0	0.22917	1
2	0	0.01439	0
3	1	0.22917	1
4	0	0.01439	0
5	1	0.93750	1
6	0	0.01439	0
7	0	0.00000	0
8	0	0.01439	0
9	0	0.75000	1
10	1	0.42308	1
11	0	0.42308	1
12	0	0.00000	0
13	0	0.01439	0
14	1	0.42308	1
15	1	0.00000	0
16	0	0.01439	0
17	0	0.22917	1
18	0	0.01439	0
19	1	0.42308	1
20	0	0.01439	0
21	0	0.01439	0
22	0	0.01439	0
23	0	0.42308	1
24	1	0.93750	1



Eine weitere Metrik der Modellevaluation entsteht auf Basis der ROC-Kurve: **Area Under the Curve (AUC)**.

```
metrics.roc_auc_score(y_true = y_test, y_score = tree_predictions_proba[:,1])
```

Code Session – Risiko Modellierung und ROC Kurve

Klassifikation – Multi-Class Evaluation

Das **Hauptproblem der Multi-Class Evaluation** liegt darin, dass eine **einzelne Metrik** wie Accuracy, Precision oder F1-Score oft nicht die gesamte Modellleistung widerspiegelt. Zwei Gründe sind dabei ausschlaggebend:

- **Unbalancierte Klassen:** Ungleichgewicht zwischen den Klassen kann die Evaluierung verzerrn.
- **Mehrdeutigkeit in den Vorhersagen:** Das Modell kann für einige Instanzen mehrere plausible Klassen vorhersagen.

Deshalb ist es entscheidend, Metriken wie **Makro-F1-Score**, **Präzision** und **Recall pro Klasse (Mikro- und Makro Average)** zu berücksichtigen, um ein vollständiges Bild der Modellleistung zu erhalten.

	Actual: A	Actual: B	Actual: C	Summe
Predicted: A	50	3	1	54
Predicted: B	5	45	3	53
Predicted: C	2	4	40	46
Summe	57	52	44	153

- **Klasse A** hat mehr Instanzen als **Klasse C**, was die **Verteilung** der Daten verzerrt. Bei der Evaluierung könnte das Modell **sehr gute Ergebnisse für die größere Klasse (A)** liefern, aber **schlechtere Ergebnisse** für kleinere Klassen (z. B. C) haben, was die Gesamtbewertung beeinflusst.

- **Klassen A und B** haben vermutlich ähnliche Merkmale, die das Modell verwirren, sodass es **Instanzen von A als B** und umgekehrt klassifiziert. Diese **Fehler zwischen benachbarten Klassen** beeinträchtigen die **Präzision** und **Recall** sowohl für A als auch für B.

Modelevaluation – Mikro- und Makrodurchschnitt

Für unbalancierte Klassen bietet es sich an, den Mikro- bzw. Makrodurchschnitt der Zielmetriken wie Präzision- oder Sensitivität/Recall zu berechnen.

- **Mikro-Average** für den gesamtübergreifenden Erfolg des Modells, ohne die Klassengrößen zu berücksichtigen. Verzerrungen, wenn einige Klassen viel größer oder häufiger sind als andere.
- **Makro-Average** wenn alle Klassen gleichwertig behandelt werden, besonders bei unausgeglichenen Datensätzen. Gefahr die Leistung kleinerer Klassen überzubewerten.

Beispiel für Präzision:

Truth \ Predicted	0	1	All
0	11674	1969	13643
1	3045	2032	5077
All	14719	4001	18720

$$\text{Micro Precision} = \frac{\text{Total True Positives}}{\text{Total True Positives} + \text{Total False Positives}}$$

$$\text{Micro Precision} = \frac{13706}{13706 + 5014} = \frac{13706}{18720} \approx 0.732$$

$$\text{Precision}_0 = \frac{TP_0}{TP_0 + FP_0} = \frac{11674}{11674 + 1969} = \frac{11674}{13643} \approx 0.855$$

$$\text{Precision}_1 = \frac{TP_1}{TP_1 + FP_1} = \frac{2032}{2032 + 3045} = \frac{2032}{5077} \approx 0.401$$

$$\text{Macro Precision} = \frac{\text{Precision}_0 + \text{Precision}_1}{2} = \frac{0.855 + 0.401}{2} = \frac{1.256}{2} = 0.628$$



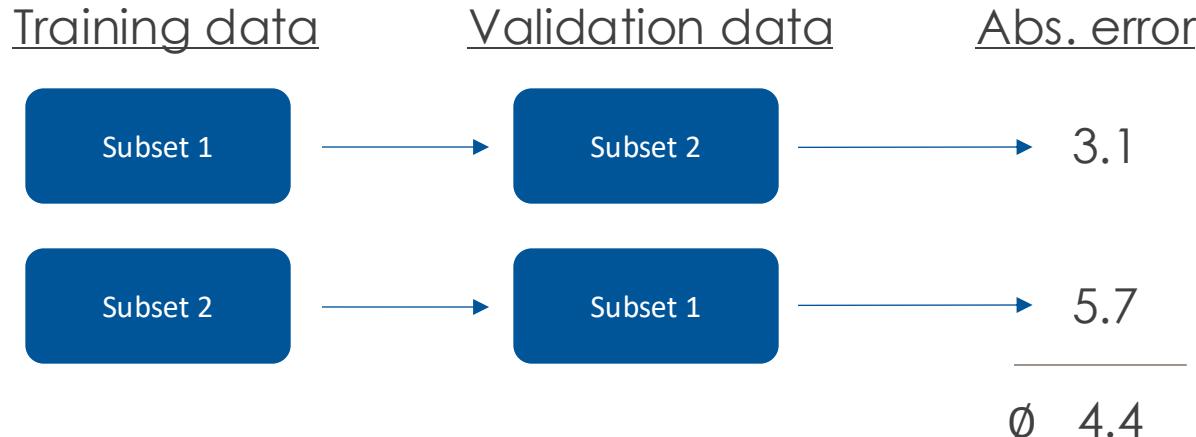
Code Session: Evaluation Wetter Prediction

Übung

Evaluiere die Vorhersage der Fussballergebnisse. Wie ausgewogen sind die Klassen? Welchen Recall und welche Precision erhältst du?

Einfache Kreuzvalidierung

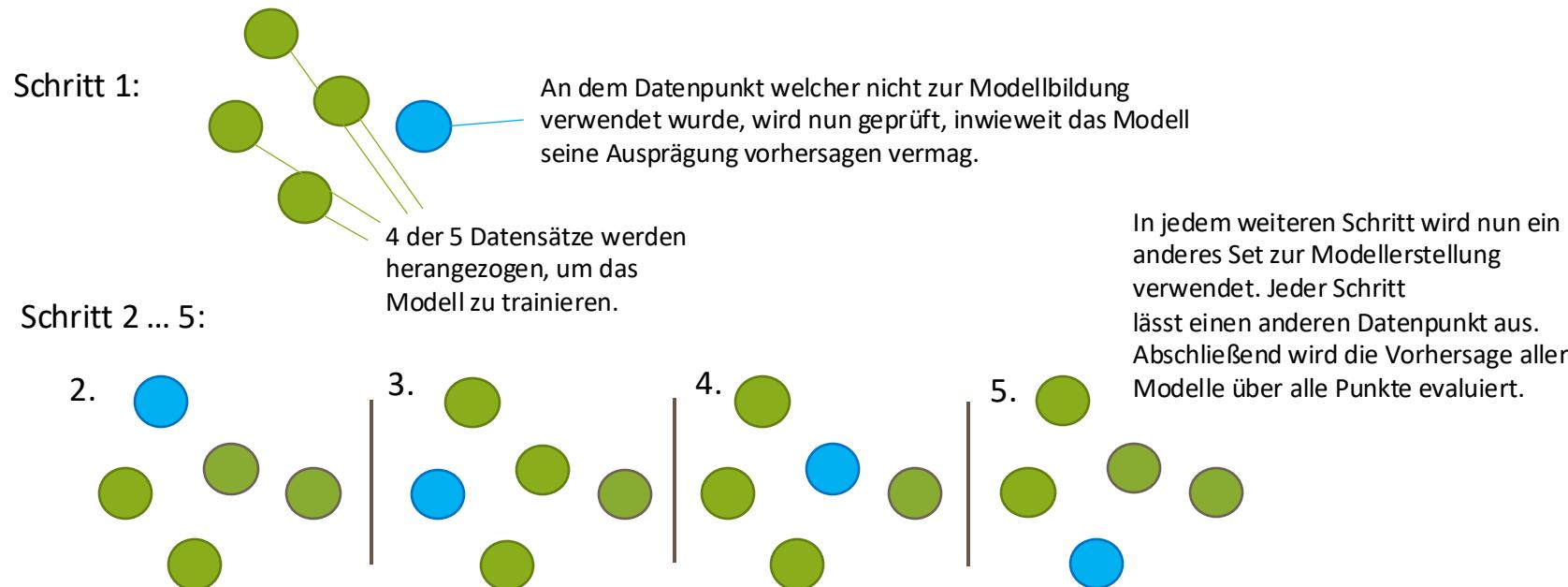
Im einfachsten Fall der **Kreuzvalidierung** werden die Daten in 2 Stichproben mit jeweils 50% der Fälle aufgeteilt. Eine dieser Stichproben wird verwendet, um das **Modell zu erstellen** (zu trainieren). Auf die andere Stichprobe wird der **Algorithmus appliziert** und der **absolute Fehler** ermittelt. Nun werden die **Rollen der Stichproben vertauscht** und wieder der absolute Fehler berechnet. Der **Mittelwert** dieser beider Fehler gibt eine Schätzung dafür, wie gut das Modell „performat“.



Eine Variante der Kreuzvalidierung ist die sog. k-Fold-Cross Validation. Dabei werden die Daten in k-Teilmengen zerlegt. Stets unter Auslassung einer dieser Teilmengen werden nun k Modelle erstellt. An der ausgelassenen Teilmenge wird das Modell schließlich validiert.

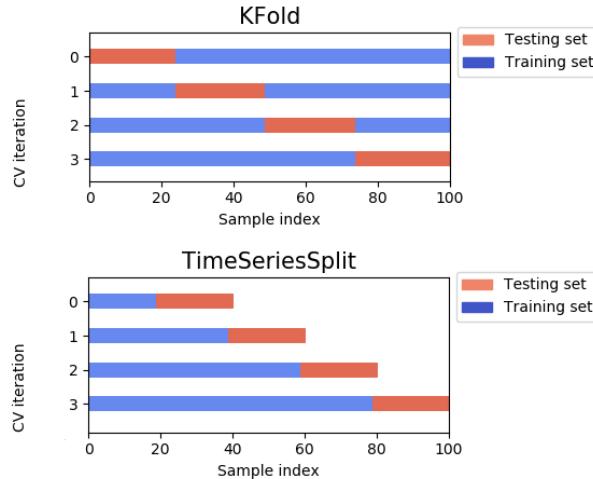
K-Fold Kreuzvalidierung

Eine Variante der Kreuzvalidierung ist die sog. **k-Fold-Cross Validation**. Dabei werden die Daten in **k-Teilmengen** zerlegt. Stets unter Auslassung einer dieser Teilmengen werden nun k Modelle erstellt. An der ausgelassenen **Teilmenge** wird das Modell schließlich **validiert**. Dies kann soweit erfolgen, dass alle Datenpunkte zur Modellbildung herangezogen werden, bis auf **einen Datenpunkt**, an dem das Modell schließlich **getestet** wird. Bei dieser sog. **Leave-One-Out** Methode entstehen ebenso viele Modelle wie Datensätze vorliegen.

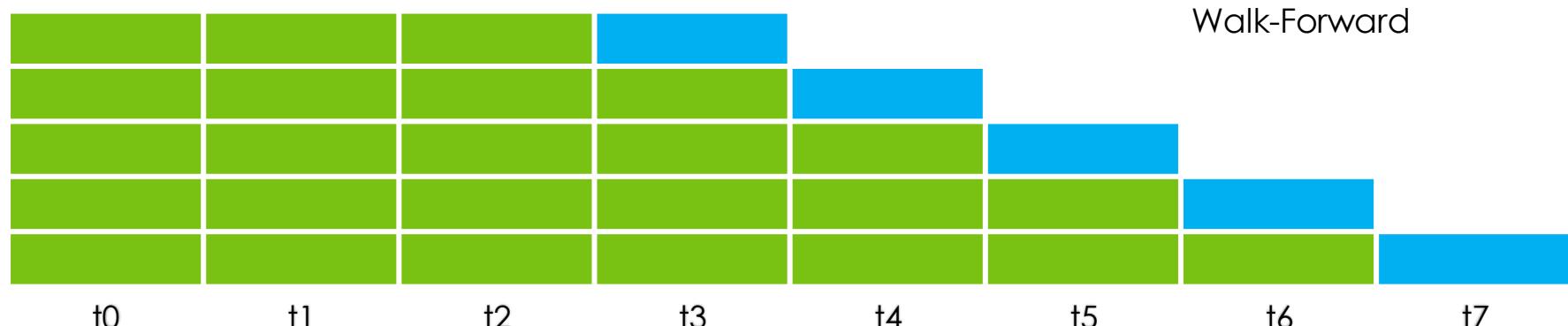
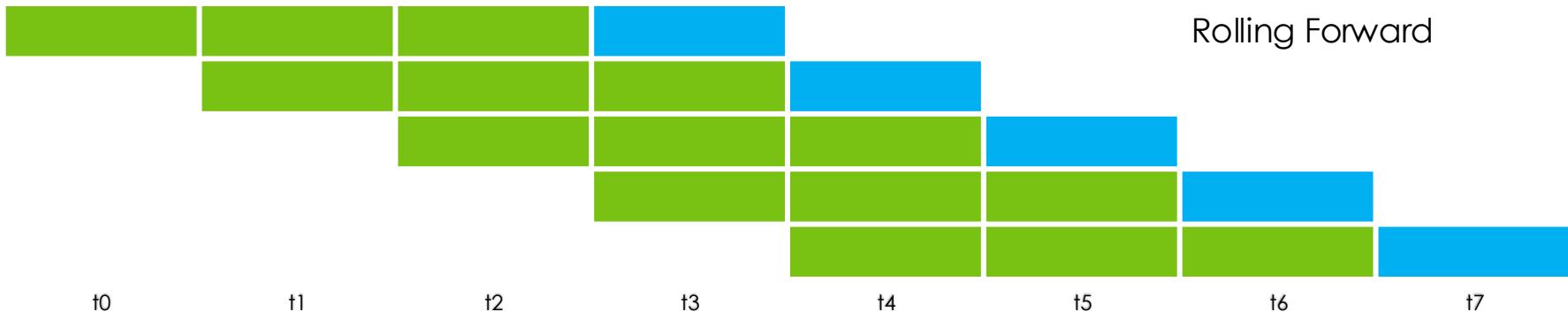


Kreuzvalidierung für Zeitreihen

Ein zentraler Unterschied von Zeitreihenanalysen zu nicht sortierten Daten besteht in der Form der Kreuzvalidierung. Die Abb. zeigt, dass die Zeitreihenkreuzvalidierung niemals den ersten Zeitpunkt in die Testdatenmenge einbezieht und niemals den letzten Zeitpunkt in die Trainingsdatenmenge.



Rolling Forward vs. Walk-Forward Validation



Code Session: Evaluation Wetter Vorhersage mit Kreuzvalidierung

Übersicht Validierungstechniken

Methode	Vorteile	Nachteile
3-Split-Validierung	- Es wird das tatsächliche Modell evaluiert. - Schnelle Anwendung	- Wenig robuste Schätzung - Benötigt große Datenmengen - Nicht alle Daten werden zum Trainieren des Modells genutzt (pessimistische Schätzung der Performance).
Einfache Kreuzvalidierung	- Einfachheit in der Modellierung	- Gefahr der gegenseitigen Lernens der beiden Teilmengen während des Parametertunings - Nicht das eigentliche Modell wird geschätzt
K-Fold-Kreuzvalidierung	- Rel. robuste Schätzung je nach Anzahl der Folds - Rechenintensität über Anzahl der Folds aussteuerbar.	- Nicht das eigentliche Modell wird geschätzt
Leave-One-Out	- Robuste Schätzung	- Rechenintensiv - Nicht das eigentliche Modell wird geschätzt. - Keine Stratifikation möglich

Kreuztabellierung

Das zentrale Werkzeug um Modellevaluationen bei Klassifikationsproblemen durchzuführen ist die Kreuztabelle. Auch wenn eine Funktion `confusion_matrix` in `sklearn` existiert um Kreuztabellen zu erstellen, bietet `pandas` mit der Funktion `crosstab` eine wesentliche flexiblere und elegantere Form der Tabellenerstellung:

```
# --- Erstelle Kreuztabelle
pd.crosstab(df_vali.Actual,
             df_vali.Class_Prediction,
             rownames=['True'],
             colnames=['Predicted'],
             margins=True)

Predicted      0     1   All
True
0              64    27    91
1               5    14    19
All            69    41   110
```

Klassifikation - Modellevaluation in sklearn

Die Funktion **classification_report** aus dem Modul **metrics** gibt einige zentrale Kennzahlen des Modells aus.

```
from sklearn import metrics
tree_report = metrics.classification_report(y_test,
                                              tree_predictions,
                                              labels = [1,0],
                                              target_names = ['Yes', 'No'])
print(tree_report)
```

	precision	recall	f1-score	support	
Yes	0.70	0.37	0.48	19	Der Anteil der echt Positiv-Werte an den positiv prognostizierten Werten insgesamt
No	0.88	0.97	0.92	91	Der Anteil der echt Positiv-Werte an den tatsächlich positiven Werten.
avg / total	0.85	0.86	0.85	110	F1 Score ist der gewichtete Durchschnitt von Precision and Recall. Bei dieser Bewertung werden sowohl falsch-positive als auch falsch-negative Ergebnisse berücksichtigt. Der Support gibt an, wie viele prognostizierte Fälle in dieser Klasse liegen.

Kreuzvalidierung

Für die Evaluation von Modellen und zur Auswahl der Modellparameter stehen in sklearn (Modul 'model_selection') Methoden zur Kreuzvalidierung zur Verfügung. Die beiden zentralen Funktionen sind `cross_validate` und `cross_val_score`. Die 2 Funktionen unterscheiden in 2 Aspekten:

1. `cross_validate` returniert ein Dictionary, `cross_val_score` ein array
2. `cross_validate` kann mehrere scorer in einem Aufruf verarbeiten

Der Parameter kann mit einem Integer spezifiziert werden – dieser repräsentiert die Anzahl von Folds – oder mit einem der in der Tabelle aufgeführten Cross-Validation-Generatoren.

```
cross_validate(cls,
               X=x,
               y=y,
               cv=5, # Amount of folds or cv-generator
               scoring=['accuracy', 'f1_macro', 'f1_weighted'],
               return_train_score=True) # Return the result
```

```
cross_val_score(cls, x, y, cv=5, scoring='f1_macro')
```

Cross-validation generators

<code>KFold (n_splits, shuffle, random_state)</code>	<code>StratifiedKFold (n_splits, shuffle, random_state)</code>	<code>GroupKFold (n_splits)</code>
Splits it into K folds, trains on K-1 and then tests on the left-out.	Same as K-Fold but preserves the class distribution within each fold.	Ensures that the same group is not in both testing and training sets.
<code>ShuffleSplit (n_splits, test_size, train_size, random_state)</code>	<code>StratifiedShuffleSplit</code>	<code>GroupShuffleSplit</code>
Generates train/test indices based on random permutation.	Same as shuffle split but preserves the class distribution within each iteration.	Ensures that the same group is not in both testing and training sets.
<code>LeaveOneGroupOut ()</code>	<code>LeaveGroupsOut (n_groups)</code>	<code>LeaveOneOut ()</code>
Takes a group array to group observations.	Leave P groups out.	Leave one observation out.
<code>LeavePOut (p)</code>	<code>PredefinedSplit</code>	
Leave P observations out.	Generates train/test indices based on predefined splits.	

Modelevaluation - Scorer

Die Modelevaluation muss in Abhängigkeit zur Fragestellung der Analyse aufgrund spezifischer Metriken erfolgen. In der Tabelle befinden sich alle zur Verfügung stehenden Metriken. Diese können als eigene Funktion genutzt werden, oder als String in Funktionen übergeben werden.

Scoring	Function	Comment
Classification		
'accuracy'	metrics.accuracy_score	
'balanced_accuracy'	metrics.balanced_accuracy_score	for binary targets
'average_precision'	metrics.average_precision_score	
'brier_score_loss'	metrics.brier_score_loss	
'f1'	metrics.f1_score	
'f1_micro'	metrics.f1_score	micro-averaged
'f1_macro'	metrics.f1_score	macro-averaged
'f1_weighted'	metrics.f1_score	weighted average
'f1_samples'	metrics.f1_score	by multilabel sample
'neg_log_loss'	metrics.log_loss	requires predict_proba support
'precision' etc.	metrics.precision_score	suffixes apply as with 'f1'
'recall' etc.	metrics.recall_score	suffixes apply as with 'f1'
'roc_auc'	metrics.roc_auc_score	
Clustering		
'adjusted_mutual_info_score'	metrics.adjusted_mutual_info_score	
'adjusted_rand_score'	metrics.adjusted_rand_score	
'completeness_score'	metrics.completeness_score	
'fowlkes_mallows_score'	metrics.fowlkes_mallows_score	
'homogeneity_score'	metrics.homogeneity_score	
'mutual_info_score'	metrics.mutual_info_score	
'normalized_mutual_info_score'	metrics.normalized_mutual_info_score	
'v_measure_score'	metrics.v_measure_score	
Regression		
'explained_variance'	metrics.explained_variance_score	
'neg_mean_absolute_error'	metrics.mean_absolute_error	
'neg_mean_squared_error'	metrics.mean_squared_error	
'neg_mean_squared_log_error'	metrics.mean_squared_log_error	
'neg_median_absolute_error'	metrics.median_absolute_error	
'r2'	metrics.r2_score	

Inhaltlicher Ablauf

Tag	Thema
1	Einführung Pandas / Numpy
	Einführung in das Machine Learning
	Datenmanagement für Machine Learning
	Erste Prognose für den Machine Learning Task
	Vorstellung der Übungstasks
2	Modellevaluation
	Anwenden unterschiedlicher Modelle
	Ensemble Modellierung
3	Anwenden unterschiedlicher Modelle auf Übungstasks
	Automatisierte Parametersuche
	Regression
	Clusteranalyse
	Neuronale Netze
	Ausblick

The screenshot shows the official scikit-learn website. At the top, there's a navigation bar with links for Home, Installation, Documentation, Examples, Google Custom Search, and a search icon. A "Fork me on GitHub" button is located in the top right corner. The main content area features a large blue banner with the text "scikit-learn" and "Machine Learning in Python". Below the banner is a grid of 25 small plots illustrating various machine learning concepts like classification, regression, and clustering. To the right of the banner is a list of bullet points highlighting the library's features: Simple and efficient tools for data mining and data analysis, Accessible to everybody, and reusable in various contexts, Built on NumPy, SciPy, and matplotlib, and Open source, commercially usable - BSD license.

Classification
Identifying to which category an object belongs to.
Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, ...
— Examples

Regression
Predicting a continuous-valued attribute associated with an object.
Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, ...
— Examples

Clustering
Automatic grouping of similar objects into sets.
Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: K-Means, spectral clustering, mean-shift, ...
— Examples

Dimensionality reduction
Reducing the number of random variables to consider.
Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization.
— Examples

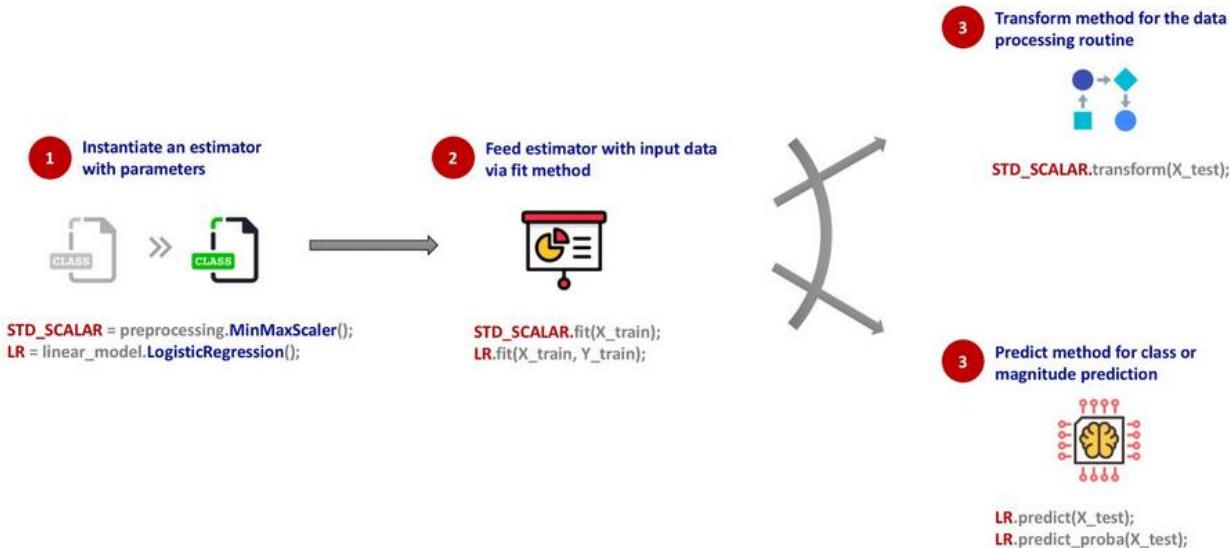
Model selection
Comparing, validating and choosing parameters and models.
Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics.
— Examples

Preprocessing
Feature extraction and normalization.
Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction.
— Examples

Transformer und Estimator

In scikit-learn existieren 2 zentrale Typen von Klassen: **Transformer** und **Estimator** Klassen.

1. **Transformer**: Verfügen über die Methoden **fit()** und **transform()**. Transformer werden zur Datenaufbereitung verwendet.
2. **Estimator**: Verfügen über die Methoden **fit()** und **predict()**. Estimator-Klassen bilden mit der Prognose (häufig) den Abschluss der Analyse.



Code Session

Übung

Verwende einen beliebigen Algorithmus aus der scikit-learn Bibliothek und variiere die Parameter. Wie ist die Performance im Vergleich?

Inhaltlicher Ablauf

Tag	Thema
1	Einführung Pandas / Numpy
	Einführung in das Machine Learning
	Datenmanagement für Machine Learning
	Erste Prognose für den Machine Learning Task
	Vorstellung der Übungstasks
2	Modellevaluation
	Anwenden unterschiedlicher Modelle
	Ensemble Modellierung
	Anwenden unterschiedlicher Modelle auf Übungstasks
3	Automatisierte Parametersuche
	Regression
	Clusteranalyse
	Neuronale Netze
	Ausblick

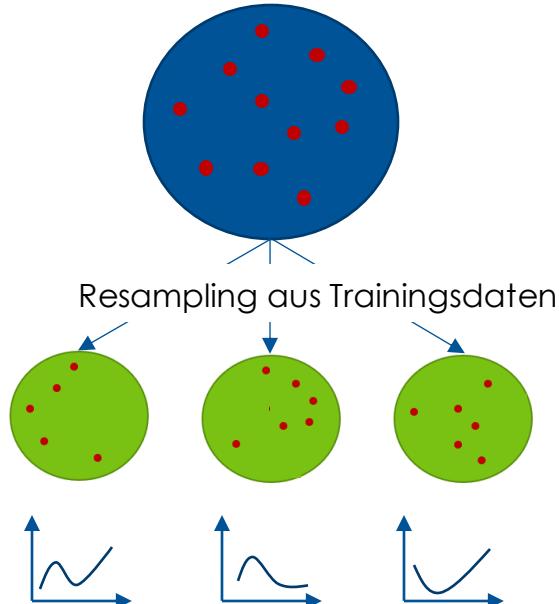
- Entscheidungs – und Regressionsbäume (**einfache Lerner**)
- Ensemble Verfahren
 - **Bagging (Parallele Modellierung)**
 - Einfaches Bagging (zufallsbasierte Fallauswahl)
 - Random Forest (zufallsbasierte Fall- und Variablenauswahl)
 - **Boosting (Sequenzielle Modellierung)**
 - Adaptive Boosting (kurz: Ada)
 - Gradient Boosting
 - XG-Boost

Bagging und Boosting - Ensembling (viele Lernen arbeiten zusammen!)

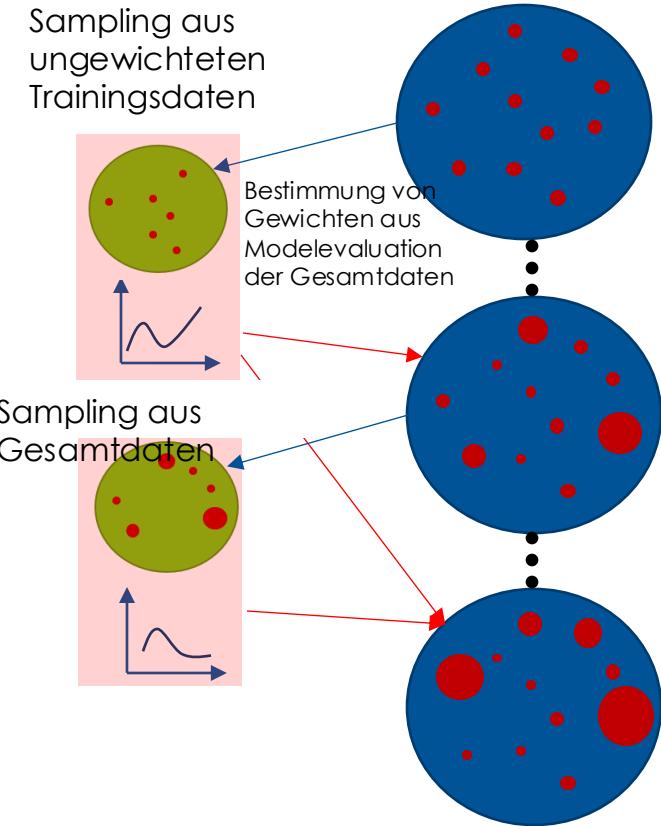
Einfacher Lerner



Bagging



Boosting



Bagging mit Random Forest

Ansatz / Idee:

Die Random Forrest Methode basiert auf Entscheidungs- und Regressionsbäumen und fügt diese zu einem Ensemble zusammen.

Statt einen einzelnen Baum zu bilden, wird ein ganzer Wald von Entscheidungsbäumen mit zufällig ausgewählten Fällen und Variablen erstellt. Das Gesamtergebnis mittelt die Ausgabewerte der einzelnen Bäume.



Vorteile:

■ **Ausbalanciert**

Einzelne Entscheidungs- und Regressionsbäume sind häufig entweder zu einfach (wenig akkurat) oder zu speziell (overfitting). Random Forest findet besser die Mitte zwischen Over- und Underfitting.

■ **Robust**

RF reagieren nur wenig auf kleinere Änderungen der Trainingsdaten. Sie sind außerdem unsensibel für Ausreißer und verzerrte Verteilungen.

■ **Random Forest erkennt nicht-lineare Beziehungen**

■ **Gute Prognoseergebnisse**

Die Prognoseergebnisse von sind besser als die von einzelnen Bäumen.

■ **Übersicht über Feature Importance**

■ **Geringes Datenmanagement**

Normalisierung der Daten nicht Notwendig.

■ **Parallelisierbarkeit**

Nachteile:

■ Nach dem Training ist das RF-Model **relativ langsam** (die vielen Bäume müssen gerechnet werden)

■ **RF ist nicht in der Lage zu extrapolieren**

■ Es existieren andere, **mächtigere Modeltypen**.

Bagging (Einfache Bootstrap Aggregation)

Bagging ist eine Ensemble Methode, mit der durch Resampling eine Vielzahl von Datensätzen erzeugt werden, auf denen jeweils ein Model trainiert wird.

Age	Sex	Weigth	Heart-Attack
55	m	105	yes
68	m	98	yes
71	f	56	no
52	m	88	no
63	m	75	yes
82	f	85	yes

Sample 1

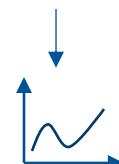
Age	Sex	Weigth	Heart-Attack
55	m	105	yes
55	m	105	yes
63	m	75	yes
55	m	105	yes
63	m	75	yes
71	f	56	no

Sample 2

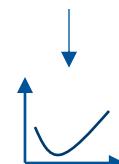
Age	Sex	Weigth	Heart-Attack
68	m	98	yes
68	m	98	yes
71	f	56	no
52	m	88	no
63	m	75	yes
52	m	88	no

Sample 3

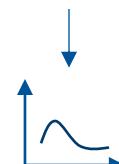
Age	Sex	Weigth	Heart-Attack
55	m	105	yes
68	m	98	yes
71	f	56	no
52	m	88	no
63	m	75	yes
82	f	85	yes



Mod. 1



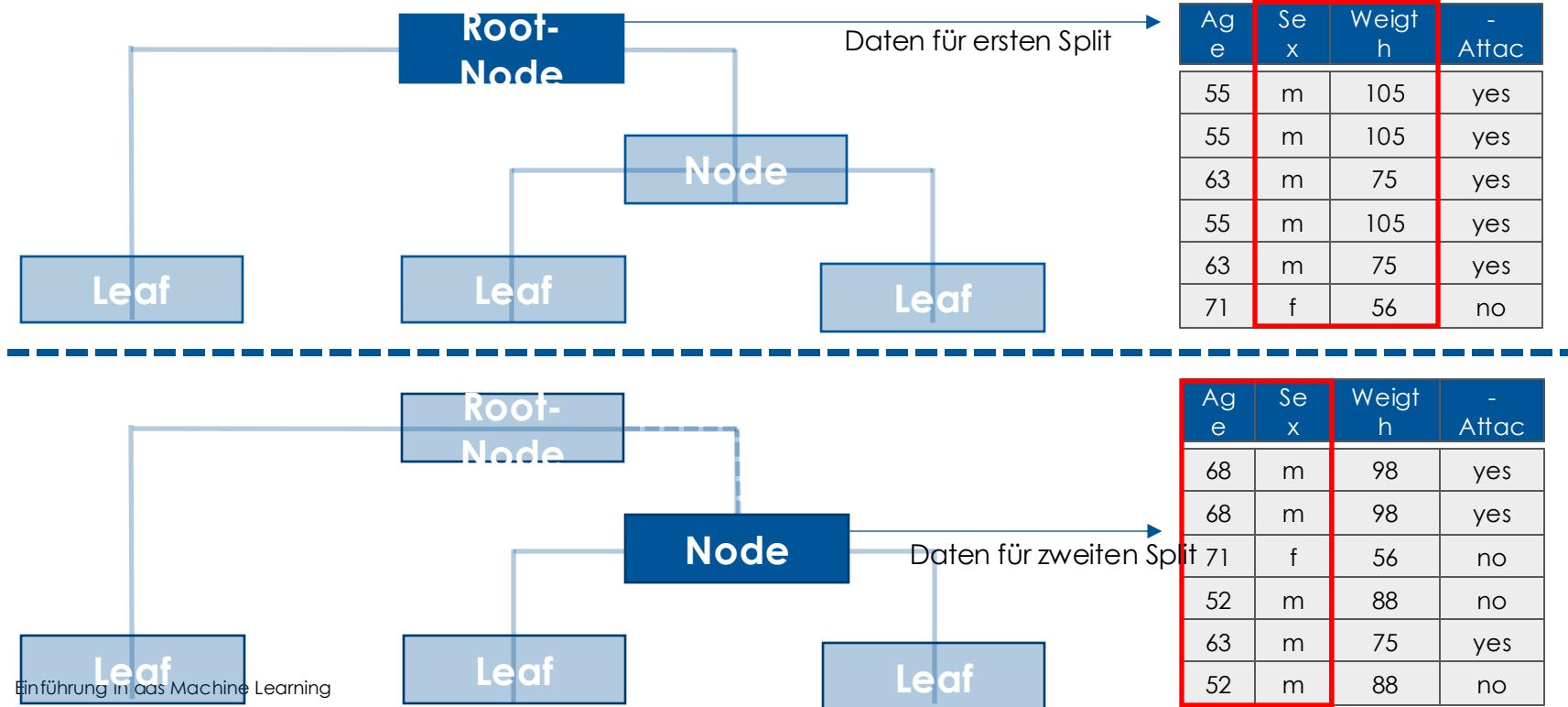
Mod. 2



Mod. 3

Random Forest – Bagging und Variablen-Sampling

Der Random Forest Algorithmus nutzt das Prinzip des Baggings und erweitert es um einen weiteren Zufallsprozess. In jedem Knoten eines Baumes wird Resampling betrieben. **Außerdem wird in jedem Knoten ein Subset von Variablen gezogen.**



Random Forest – Out of bag error

Machine Learning Verfahren die intern Bootstrap-Samples verwenden, können über den sog. Out of Bag-Error evaluiert werden. Dazu wird zunächst das Model anhand der Bootstrap-Daten erstellt. Anschließend werden alle Fälle (die nicht Teil der Trainingsdaten waren) dazu verwendet, den Vorhersagefehler zu ermitteln. Wird das Argument oob_score in der Klasse RandomForestClassifier auf True gesetzt, kann die darüber ermittelte Accuracy als Alternative zur Kreuzvalidierung genutzt werden.

Age	Sex	Weight	-	Attack
55	m	105		yes
68	m	98		yes
71	f	56		no
52	m	88		no
63	m	75		yes
82	f	85		yes

Bootstrap-Sample

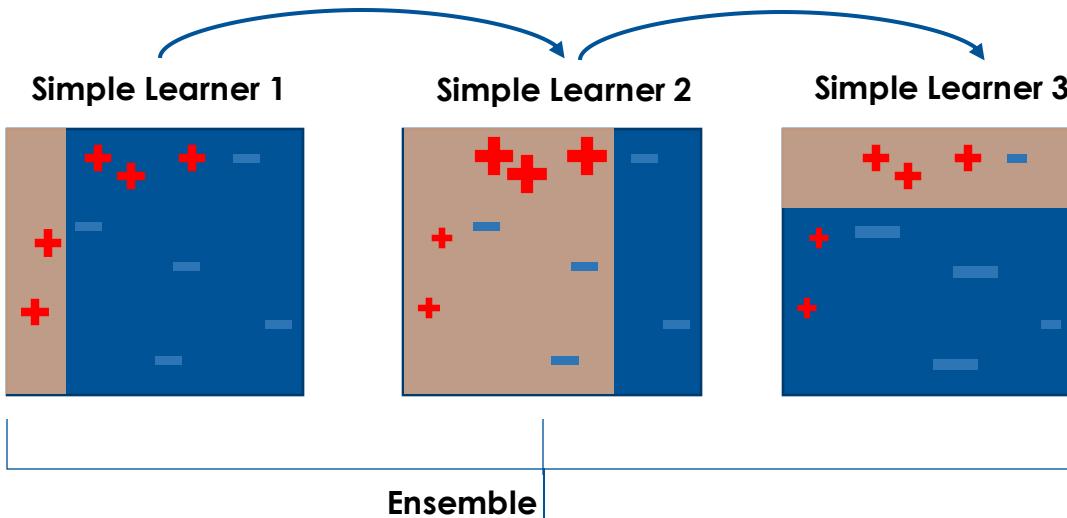
Age	Sex	Weight	-	Attack
55	m	105		yes
55	m	105		yes
63	m	75		yes
55	m	105		yes
63	m	75		yes
71	f	56		no

Anhand der Daten des Bootstrap-Samples wird das Model erstellt.

Age	Sex	Weight	-	Attack
68	m	98		yes
52	m	88		no
82	f	85		yes

Fälle die nicht Teil des Bootstrap-Samples waren, werden zur Validierung des Models verwendet.

Boosting – Adaptive-Boosting



Diese Regionen werden von min. 2 Modellen "positiv" klassifiziert und ist somit im Ensemble "positiv".

Boosting

Die Idee des Boostings ist es, einfache Lerner durch Gewichtsveränderungen an den Trainingsdaten sukzessiv zu einem starken Lerner zu vereinen.

In einem ersten Schritt, wird ein Modell mit einem einfachen Klassifikator erstellt – in der Regel werden im ersten Schritt alle Fälle mit gleichem Gewicht berücksichtigt. Anschließend wird dieses einfache Modell evaluiert. Falsche Prognosen oder große Abweichungen vom echten Wert führen zu einem hohen Gewicht für den jeweiligen Fall. Mit der 'neuen' Gewichtung wird wiederum ein Klassifikator erstellt, der eine Gewichtsanpassung bedingt usw. Die Zusammengesetzten 'einfachen' Modelle ergeben das Gesamtmodell.

Age	Sex	Weigth	Heart-Attack
55	m	105	yes
68	m	98	yes
71	f	56	no
52	m	88	no
63	m	75	yes
82	f	85	yes

Sample-Weight
1/6
1/6
1/6
1/6
1/6
1/6

Model 1

Wenn Weight > 70

yes richtig: 4 (4)
falsch: 0 (0)

no richtig: 1 (1)
falsch: 1 (1)

Sample-Weight
0.5/6
0.5/6
3.5/6
0.5/6
0.5/6
0.5/6

Model 2

Wenn Sex == m

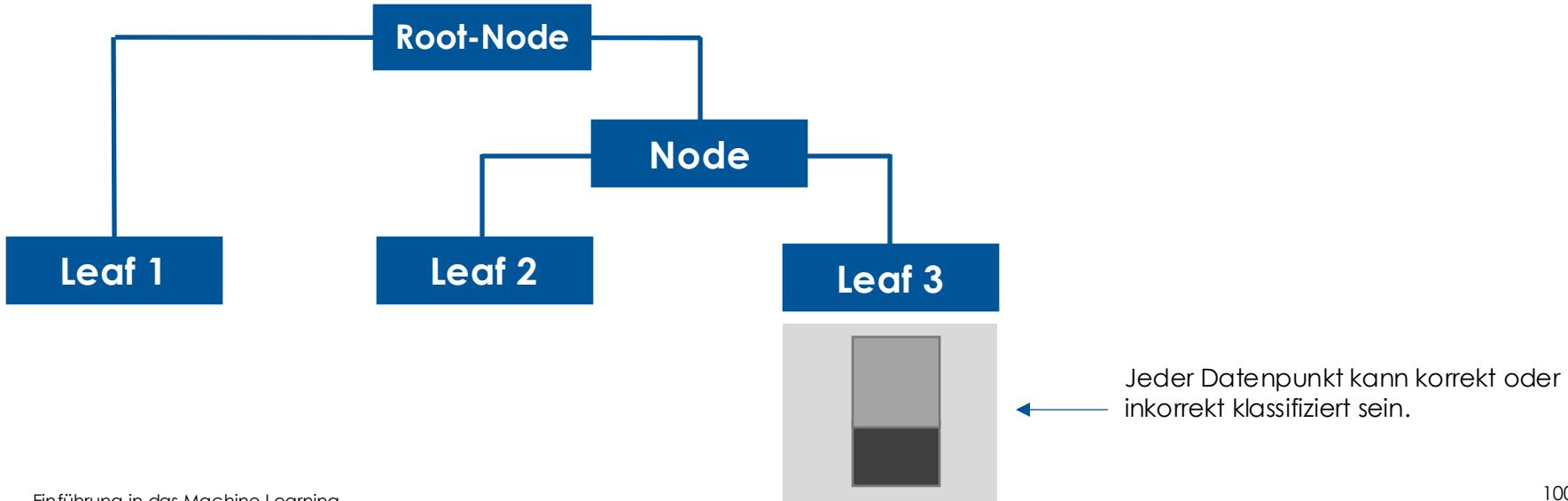
yes richtig: 3 (1.5)
falsch: 1 (0.5)

no richtig: 1 (3.5)
falsch: 1 (0.5)

Sample-Weight
0.4/6
0.4/6
2.1/6
0.4/6
0.4/6
1.3/6

Gradient Boosting für Klassifikationen

1. Erstellen eines Regressionsbaumes mit einem Gewicht von 1 für jede Observation
2. Verändere das Gewicht jeder Observation nach folgender Regel:
 1. Korrekte Klassifikation: neues_gewicht = altes_gewicht * $\exp(-1 \cdot \text{learning_rate}) / \text{summe_alle_gewichte}$
→ Das Gewicht wird reduziert!
 2. Falsche Klassifikation : neues_gewicht = altes_gewicht * $\exp(+1 \cdot \text{learning_rate}) / \text{summe_alle_gewichte}$
→ Das Gewicht wird erhöht!



Gradient Boosting für Klassifikationen – Beispiel

Angenommen sei eine Lernrate von 0.1:

- Das Gewicht von korrekt klassifizierten Fällen ändert sich mit $1/\exp(0.1) \sim=0.9$
- Das Gewicht von inkorrekt klassifizierten Fällen ändert sich mit $\exp(0.1) \sim=1.1$

Angenommen seien 1100 Observationen:

- Bei 1000 korrekt Klassifikationen verbleiben ,nur' etwa 900 ($1000 * 0.9$) im nächsten Iterationsschritt
- Bei 100 inkorrekt Klassifikationen verbleiben etwa 110 ($100 * 1.1$) im nächsten Iterationsschritt
- Die nächste Iteration verarbeitet 1010 Fälle

Die Gewichtsanpassungen und Modellierungen neuer Bäume werden fortgeführt, solange eine Verbesserung der Residuen erzielt werden kann.

Gradient Boosting - XGBoost

Eine bekannte Erweiterung des Konzepts von Gradient Boosting Machines ist der Algorithmus XGBoost (Extreme Gradient Boosting). Er enthält folgende Features:

- Randomisierung der Daten (vergleichbar mit Random Forest – einzelne Bäume sind nicht mehr so stark korreliert)
- Nutzt Regularisierung (L1 & L2)
- Parallelisierbar
- Nutzt Gradienten 2ter Ordnung. Durch die Ableitungen der Loss-Funktion ist die Richtung der Gradienten bekannt und das Minimum der Loss-Funktion wird schneller gefunden.
- Können mit Missing-Werten umgehen

Code Session Bagging und Boosting

Übung

Probiere verschiedene Modelle mit Bagging und Boosting aus. Welches funktioniert am besten? Beobachte dabei das Overfitting.

The background features abstract white line art on a dark blue background. On the left, there's a complex, fan-like structure composed of many thin lines. On the right, there are several parallel, wavy lines.

Code Session Ensemble von Modellen

Random Forest - Argumente

`class sklearn.ensemble`

```
RandomForestClassifier(  
    n_estimators='warn',  
    criterion='gini',  
    max_depth=None,  
    min_samples_split=2,  
    min_samples_leaf=1,  
    min_weight_fraction_leaf=0.0,  
    max_features='auto',  
    max_leaf_nodes=None,  
    min_impurity_decrease=0.0,  
    min_impurity_split=None,  
    bootstrap=True,  
    oob_score=False,  
    n_jobs=None,  
    random_state=None,  
    verbose=0,  
    warm_start=False,  
    class_weight=None)
```

n_estimators: Anzahl der Bäume

criterion: "gini" oder "entropy"

Argumente entsprechen Decision-Trees:

max_depth: Maximale Tiefe des Baumes (Anzahl Ebenen von Root-Node zu Leaf)

min_samples_split: Min. Anzahl von Fällen für einen weiteren Split.

min_samples_leaf: Min. Anzahl von Fällen in einem Leaf.

min_weight_fraction_leaf: Min. Summe aller Fallgewichtungen in einem Leaf.

max_features: Anzahl der Features die für einen weiteren Split betrachtet werden.

max_leaf_nodes: Max. Anzahl der Leaf-Nodes.

min_impurity_decrease: Minimale Verbesserung des Modells nach Aufteilung

min_impurity_split: Deprecated (jetzt "min_impurity_split")

oob_score: Soll die Zielmetrik mit Out-Of-Bag ermittelt werden? Anders: Sollen nur Fälle die nicht Trainingsdaten waren, zur Modellevaluation genutzt werden?

bootstrap: Sollen Bootstrap-Samples genutzt werden, um die Bäume zu erstellen? Wenn False wird der gesamte Datensatz genutzt.

n_jobs: Anzahl der Prozesse zur Modellerstellung.

random_state: Fixieren des Zufallszahlengenerators und Reproduzierbarkeit gewährleisten.

verbose: Kann die Integerwerte 0, 1 und 2 annehmen. Je höher der Wert, desto ausführlicher das Logging.

warm_start: Wenn mehrmals trainiert wird, kann mit warm_start das Parameterset des letzten Trainings verwendet werden und die Startparameter des neuen Trainings bilden – somit kann Trainingszeit gespart werden.

class_weight: Dictionary mit Klassengewichten der Form: {class_label: weight}

xgBoost - Argumente

In Python ist die meistverwendete Implementierung von xgBoost in der gleichnamigen Bibliothek zu finden.

```
from xgboost import XGBClassifier  
  
XGBClassifier(  
    learning_rate=0.3,  
    min_split_loss =0,  
    max_depth=6,  
    min_child_weight=1,  
    min_samples_leaf=1,  
    subsample=1,  
    lambda=1,  
    alpha=0,  
    scale_pos_weight=1)
```

learning_rate: Wertebereich: 0:1 Wie soll sich die Gewichtsanpassung reduzieren? Je größer, desto erfolgt keine weitere Gewichtsanpassung (Overfitting wird vermieden)

min_split_loss: Minimale Verbesserung der Lossfunktion nach Aufteilung

max_depth: Maximale Tiefe des Baumes.

min_child_weight: Minimale Summe an Gewicht in einem Knoten um einen Split durchzuführen

subsample: Sollen die Trainingsdaten vor dem Erstellen der Bäume nochmals gesampled werden, so kann die Samplegröße übergeben werden.

lambda: L2 Regularisierung der Gewichte – je größer desto weniger Overfitting

alpha: L1 Regularisierung der Gewichte – je größer desto weniger Overfitting

scale_pos_weight: Für imbalancierte Binaere Modelle entscheidend.

Formel: $\text{count}(\text{negative examples})/\text{count}(\text{Positive examples})$

Weitere unter: <https://xgboost.readthedocs.io/en/latest/parameter.html>

Inhaltlicher Ablauf

Tag	Thema
1	Einführung Pandas / Numpy
	Einführung in das Machine Learning
	Datenmanagement für Machine Learning
	Erste Prognose für den Machine Learning Task
	Vorstellung der Übungstasks
2	Modellevaluation
	Anwenden unterschiedlicher Modelle
	Ensemble Modellierung
Anwenden unterschiedlicher Modelle auf Übungstasks	
3	Automatisierte Parametersuche
	Regression
	Clusteranalyse
	Neuronale Netze
	Ausblick

Übung

Verwende die Ensemble Modellierung beim Übungstask. Wie hoch ist dein Gewinn / Verlust?

Inhaltlicher Ablauf

Tag	Thema
1	Einführung Pandas / Numpy
	Einführung in das Machine Learning
	Datenmanagement für Machine Learning
	Erste Prognose für den Machine Learning Task
	Vorstellung der Übungstasks
2	Modellevaluation
	Anwenden unterschiedlicher Modelle
	Ensemble Modellierung
	Anwenden unterschiedlicher Modelle auf Übungstasks
3	Automatisierte Parametersuche
	Regression
	Clusteranalyse
	Neuronale Netze
	Ausblick

Grid-Search - Parametertuning

Eine sehr zentrale Technik für die Entscheidung über die Modellparameter ist die sog. **Rastersuche (Grid-Search)**. Bei dieser werden mögliche Ausprägungen der Parameter im Vorfeld festgelegt und anschließend alle Kombinationen von Parameterzusammenstellungen in der Modellerstellung angewendet.

```
# --- Grid-Search
from sklearn.model_selection import GridSearchCV

tree_params = {'max_depth': [2,3,4,5],
               'min_samples_split': [3,5,8],
               'min_samples_leaf': [3,5,8],
               'max_features': [3,5,8]}

from sklearn.metrics import make_scorer, precision_score
tree_scorer = make_scorer(precision_score)

tree_gridSearch = GridSearchCV(tree_clf,
                               param_grid = tree_params,
                               scoring = tree_scorer)
tree_gridSearch.fit(x_train, y_train)

print(
    tree_gridSearch.best_params_
)

→ {'max_depth': 3, 'max_features': 8, 'min_samples_leaf': 3, 'min_samples_split': 8}

# Übersicht über mögliche Scoring-Values: http://scikit-learn.org/stable/modules/model\_evaluation.html
```

The background features abstract white line art on a dark blue background. On the left, there's a complex, fan-like structure composed of many thin, curved lines. On the right, there are several parallel, wavy lines that curve upwards.

Code Session

Übung: Titanic-Datensatz

Verschaffe dir einen Überblick über den Titanic-Datensatz.
Welche Spalten sind für eine Klassifikation sinnvoll? Welche
Spalten sollten transformiert werden?

daten/titanic.csv

Übung: Klassifikation mit Titanic-Datensatz

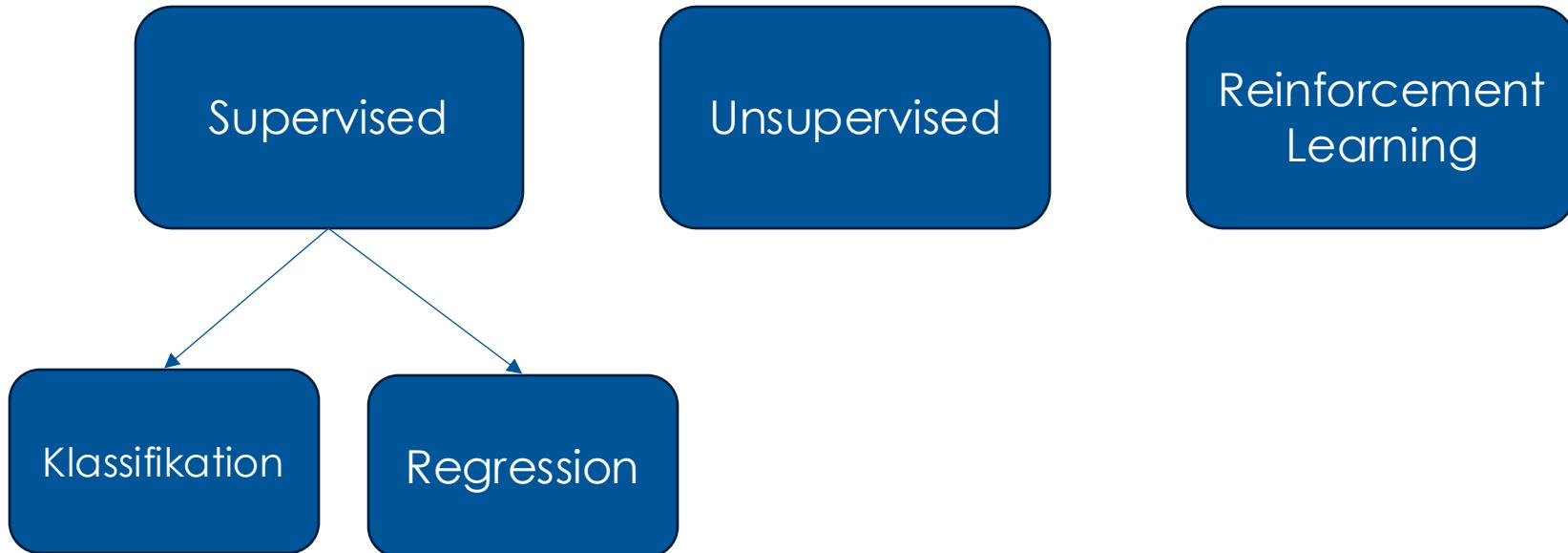
Verwende den Titanic-Datensatz für eine Klassifikation, um die Spalte „Survived“ vorherzusagen.
Nutze alle Techniken, die du kennst.
Maximiere die Accuracy.

daten/titanic.csv

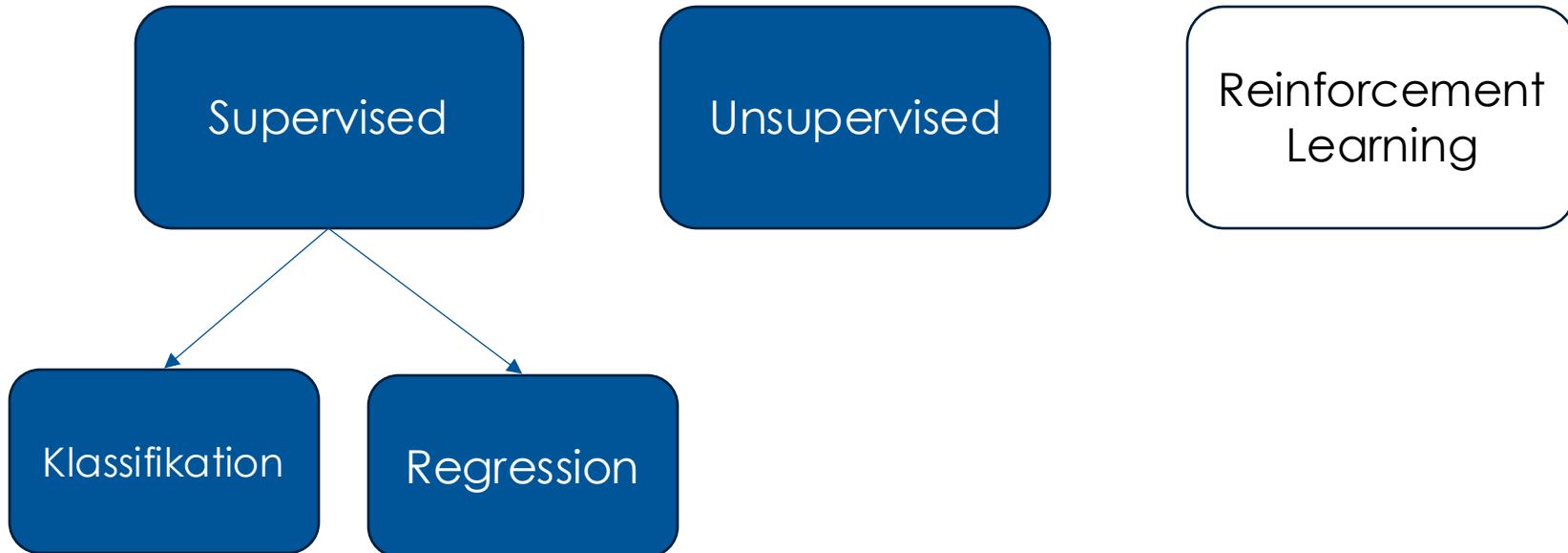
Inhaltlicher Ablauf

Tag	Thema	
1	Einführung Pandas / Numpy	
	Einführung in das Machine Learning	
	Datenmanagement für Machine Learning	
	Erste Prognose für den Machine Learning Task	
	Vorstellung der Übungstasks	
2	Modellevaluation	
	Anwenden unterschiedlicher Modelle	
	Ensemble Modellierung	
	Anwenden unterschiedlicher Modelle auf Übungstasks	
3	Automatisierte Parametersuche	
	Weitere Machine Learning Typen mit Fokus Regression	
	Clusteranalyse	
	Neuronale Netze	
	Ausblick	

Machine Learning - Problem Typen

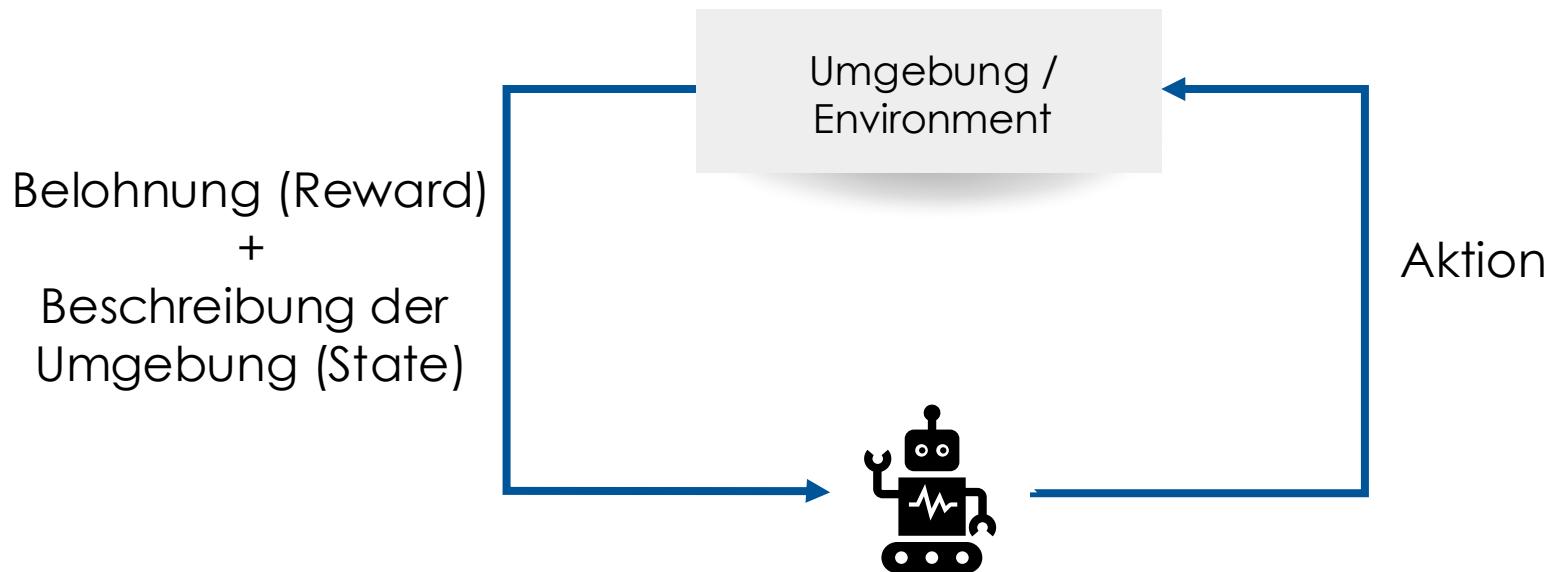


Machine Learning - Problem Typen



Reinforcement Learning

= bestärkendes Lernen



Reinforcement Learning mit Atari

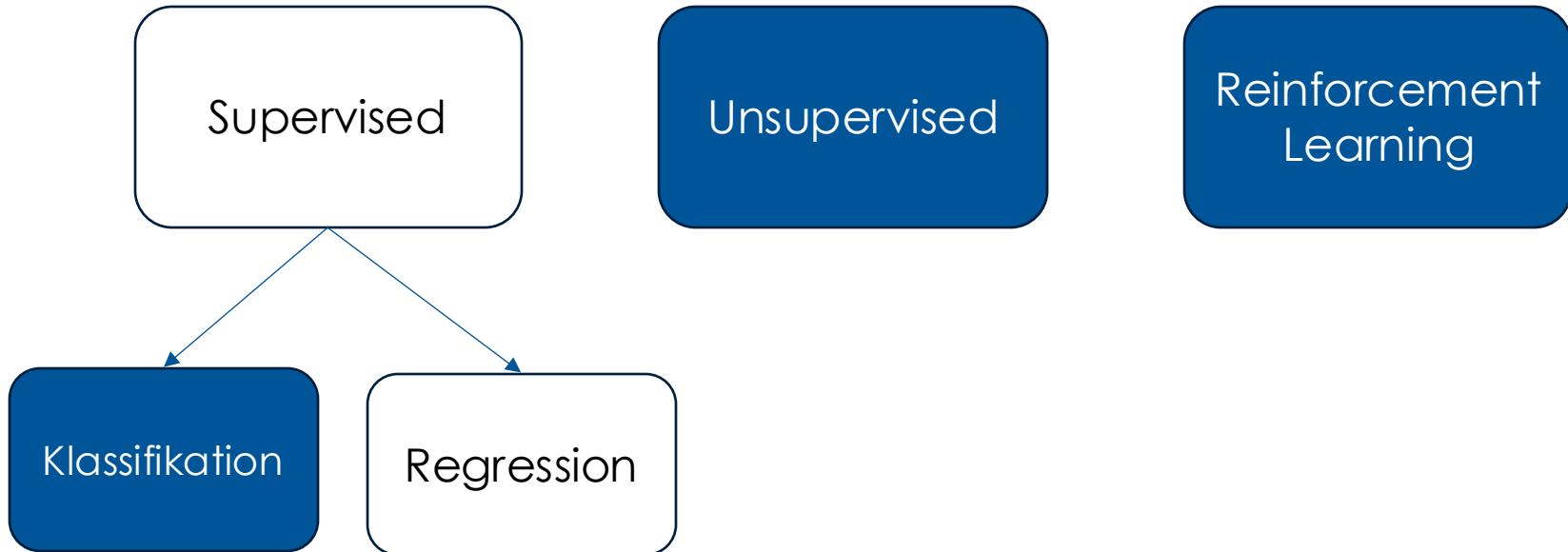


Reinforcement Learning

https://alazareva.github.io/rl_playground/

1. Setze Epsilon auf 0. Welches Verhalten zeigt der Agent?
2. Wie wirkt sich der Discount Faktor auf die Q-Werte aus?

Machine Learning - Problem Typen



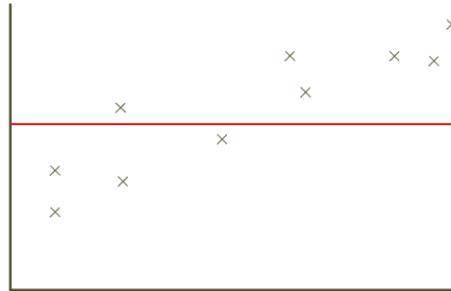
Problem type: Regression

Historische Daten

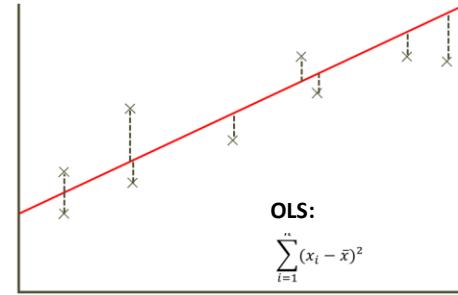


Possible models ...

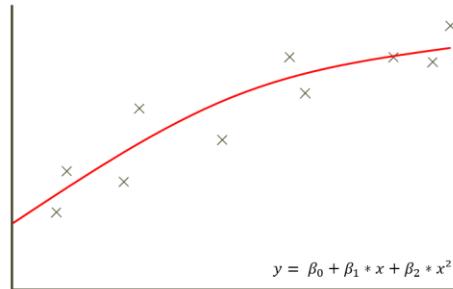
Arithmetisches Mittel



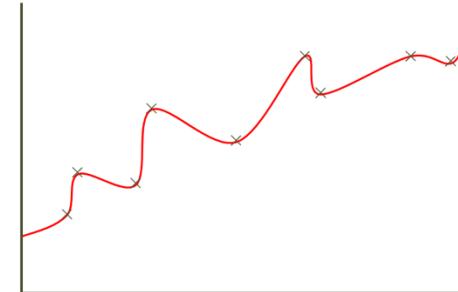
OLS-Regression



Polynom 2ten Grades



Polynom n-ten Grades

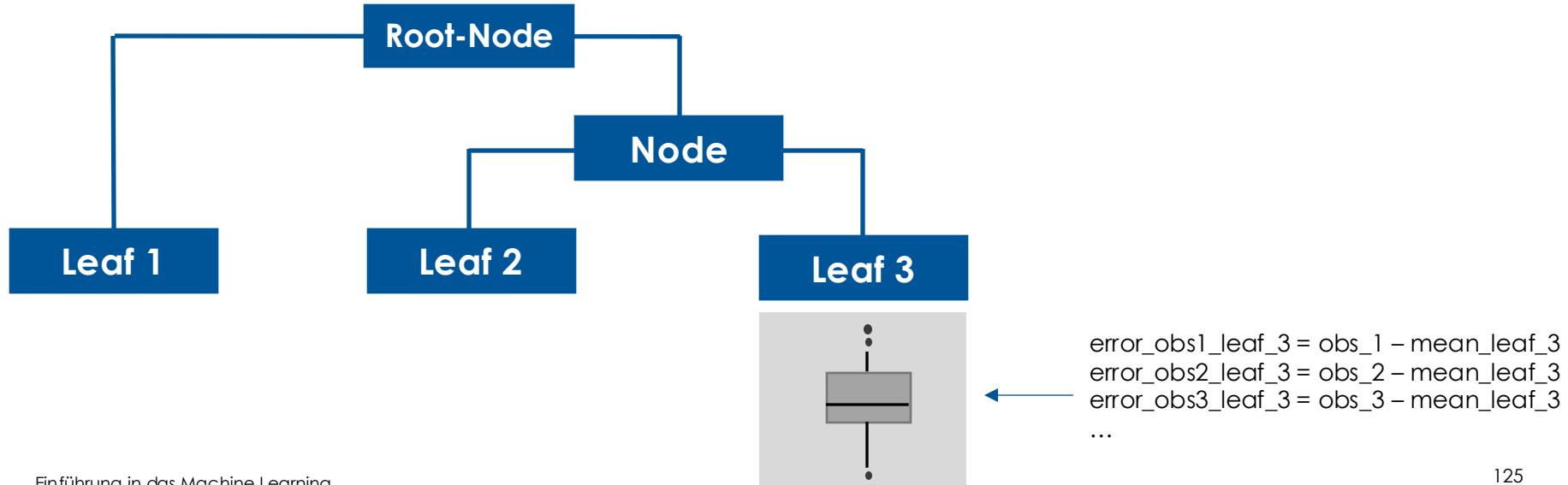


Anwendungen Regression

- Zeitreihenprognosen für Waren-Käufe
 Input: Warenbestand über die letzten Jahre
 Output: Vorhersage der Käufe in den nächsten Tagen
- Hauspreis-Bestimmung
 Input: Baujahr, Größe, Anzahl Zimmer etc.
 Output: Preis

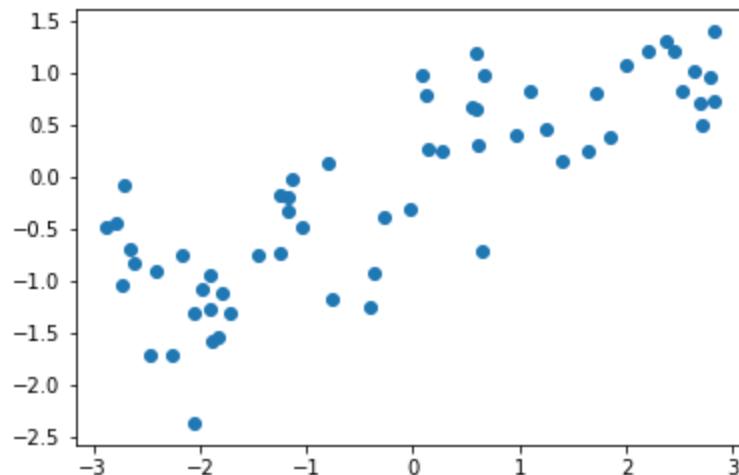
Boosting – Gradient Boosting für Regressionen

1. Erstellen eines Regressionsbaumes
2. Ermitteln der Residuen (Fehler) aller Beobachtungen in allen Leaf-Nodes
3. Erstellen des nächsten Regressionsbaumes unter Verwendung der Residuen
4. Wiederholen von Schritt 2 und 3 für eine festgelegte Anzahl von Iterationen



Regression - Modellevaluation

Die Modellevaluation von Regressionsproblemen kann zunächst anhand eines Scatterplots zwischen tatsächlichen und vorhergesagten Werten erfolgen. Je präziser die Vorhersagekraft des Modells, desto klarer ist Zusammenhang im Plot erkennbar.



Code Session Regression

Übung

Nutze folgende Mittel, die du aus der Klassifikation kennst, um die Regression zu verbessern:

- Verschiedene Algorithmen aus sklearn ausprobieren
 - Ensemble Algorithmen
 - Grid Search

Übung Autodaten

Analysiere die Autodaten mit pandas. Welche Spalten benötigen für das Machine Learning eine Transformation?

template_car_data.csv

Übung Vorhersage Kraftstoffverbrauch

Erstelle eine Regression für die Autodaten. Zielvariable ist die Spalte mpg (=miles per gallon / Kraftstoffverbrauch).
Optimiere den MSE.

template_car_data.csv

Regression - Modellevaluation

Neben der Visualisierung existieren zahlreiche Metriken, um die Modellgüte zu bewerten. In sklearn sind alle wesentlichen Metriken implementiert.

```
# --- Predictions
tree_reg_pred = tree_reg.predict(x_test)

from sklearn.metrics import explained_variance_score, r2_score, mean_absolute_error,
mean_squared_error, median_absolute_error

# --- Metriken
explained_variance_score(y_true = y_test,
                          y_pred = tree_reg_pred) → Erklärte Varianz

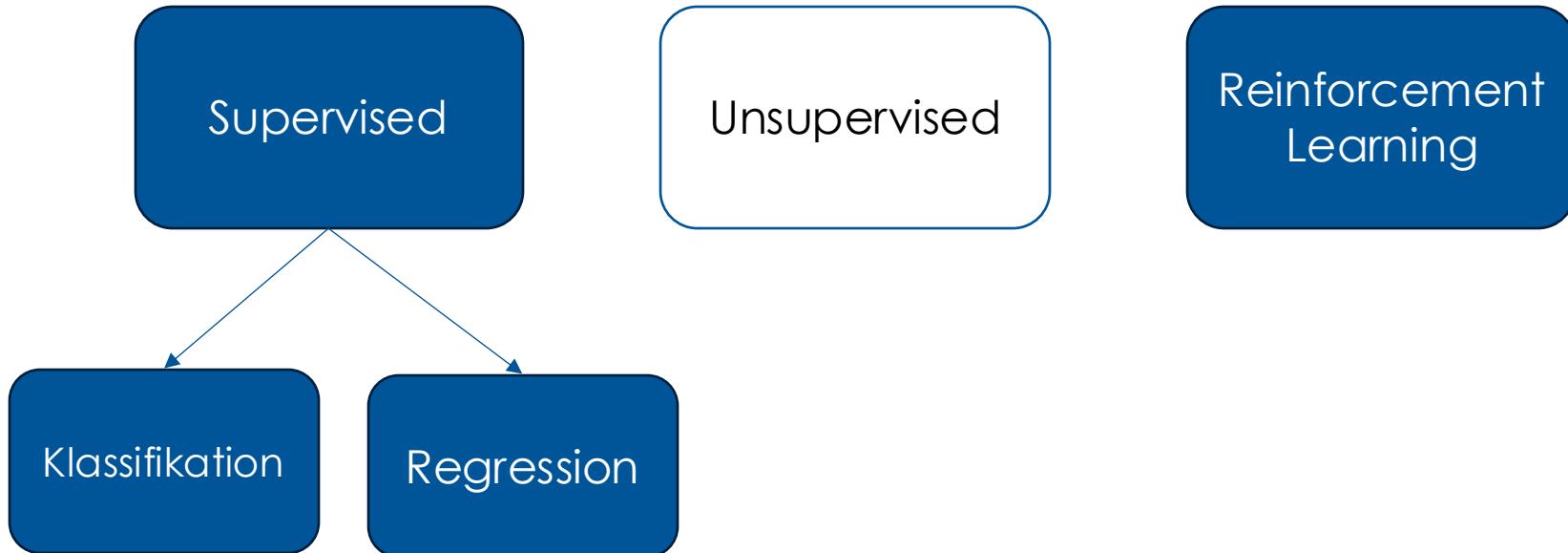
r2_score(y_true = y_test,
          y_pred = tree_reg_pred) → Korrelation zwischen Vorhersage und tatsächlichen Werten

mean_absolute_error(y_true = y_test,
                     y_pred = tree_reg_pred) → Mittlerer absoluter Fehler
mean_squared_error(y_true = y_test,
                     y_pred = tree_reg_pred) → Mittlerer quadratischer Fehler
median_absolute_error(y_true = y_test,
                      y_pred = tree_reg_pred) → Mittlerer (Median) absoluter Fehler
```

Inhaltlicher Ablauf

Tag	Thema
1	Einführung Pandas / Numpy
	Einführung in das Machine Learning
	Datenmanagement für Machine Learning
	Erste Prognose für den Machine Learning Task
	Vorstellung der Übungstasks
2	Modellevaluation
	Anwenden unterschiedlicher Modelle
	Ensemble Modellierung
	Anwenden unterschiedlicher Modelle auf Übungstasks
3	Automatisierte Parametersuche
	Regression
	Clusteranalyse
	Neuronale Netze
	Ausblick

Machine Learning - Problem Typen



Unsupervised Learning

= Unüberwachtes Lernen

Es gibt kein vorgegebenes bzw. richtiges Ergebnis.

- Cluster identifizieren
- Anomalie Erkennung
- Empfehlungsdienste (Recommender Systems)

Anwendungen Cluster Analyse

- Als Teil der Datenanalyse (Intelligenz Data Analytics)
- Verstehen der Kundengruppen
 - Input: Alter, Geschlecht, gekaufte Waren
 - ↓ Output: Einteilung der Kunden in Gruppen mit speziellen Merkmalen

Anwendungen Anomalie Erkennung

- Überwachung eines Maschinenparks um ungewöhnliches Verhalten zu erkennen
 - Input: Sensordaten
 -  Output: Ein Score, der den Grad der Abweichung von der Norm angibt.
- Überwachung der Input Daten für einen anderen Machine Learning Algorithmus um Meßfehler zu erkennen
 - Input: Wetter Daten
 -  Output: Ein Score, der den Grad der Abweichung von der Norm angibt.

Anwendungen Empfehlungssystemen

- Empfehlung für zu Kaufende Waren
 Input: Alter, Geschlecht, bisher gekaufte Waren
 Output: Liste von Waren
- Empfehlung für den nächsten Film
 Input: Alter, Geschlecht, bisher gesehene Filme
 Output: Liste von Filmen

Was ist Clusteranalyse?

- Begriffserklärung und Zielsetzung: **Gruppierung von Datenpunkten ohne vorgegebene Labels**
- Relevanz im Machine Learning: Clusteranalyse ist eine **unüberwachte** (unsupervised) Lernmethode (kein Training mit bekannten Ergebnissen)
- Abgrenzung zu Klassifikation: Bei der Klassifikation sind die Klassen bekannt, während bei der Clusteranalyse Strukturen erst entdeckt werden

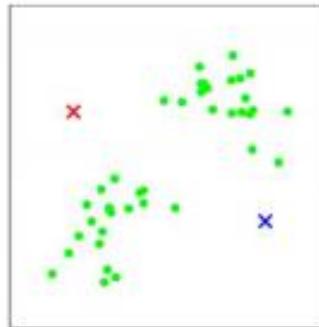
Ziele der Clusteranalyse

- **Identifikation von Mustern:** Ähnliche Datenpunkte sollen automatisch in Gruppen (Cluster) eingeteilt werden
- **Erkenntnisgewinn:** Neue Einsichten in Daten finden, z.B. Kundensegmente, Muster in Gen-Daten, Verhaltensgruppen in Logfiles
- **Informationsreduktion:** Komplexe Datensätze können durch Clusterübersicht einfacher interpretiert und visualisiert werden

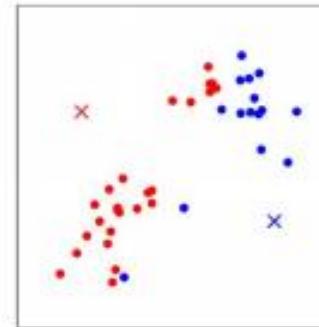
Funktionsweise von K-Means



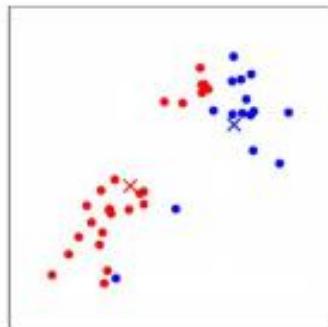
(a)



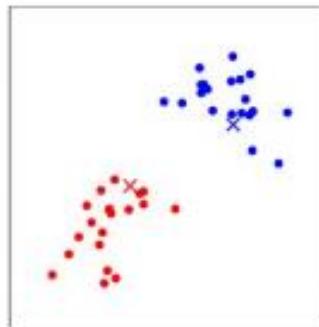
(b)



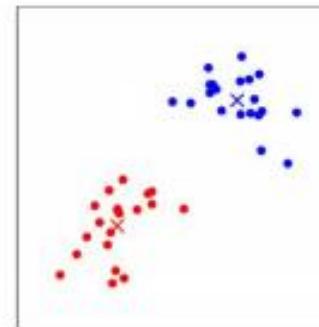
(c)



(d)



(e)



(f)

<https://stanford.edu/~cziegler/cs221/handouts/kmeans.html>

Funktionsweise von K-Means

- Auswahl der **Anzahl k an Clustern**: Zu Beginn wird festgelegt, wie viele Cluster gesucht werden
- Initialisierung: **Zufällige Platzierung der Clusterzentren** (Centroids) oder Auswahl spezieller Algorithmen zur Initialisierung
- **Iterative Zuordnung**: Datenpunkte werden dem jeweils nächstgelegenen Clusterzentrum zugeordnet und die Zentren danach neu berechnet
- **Konvergenz**: Der Prozess wiederholt sich, bis sich die Zuordnung und Clusterzentren stabilisieren oder eine Abbruchbedingung erfüllt ist

Stärken und Schwächen von K-Means

- Stärke: **Einfaches und schnelles Verfahren**, gut skalierbar bei größeren Datensätzen
- Schwäche: Erfordert die **Vorab-Festlegung von k**, was oft schwierig ist
- Anfälligkeit: Empfindlich gegenüber **Ausreißern** und nicht geeignet für unregelmäßig geformte Cluster
- Vorverarbeitung: Benötigt **standardisierte oder normalisierte Daten**, damit Distanzen aussagekräftig bleiben

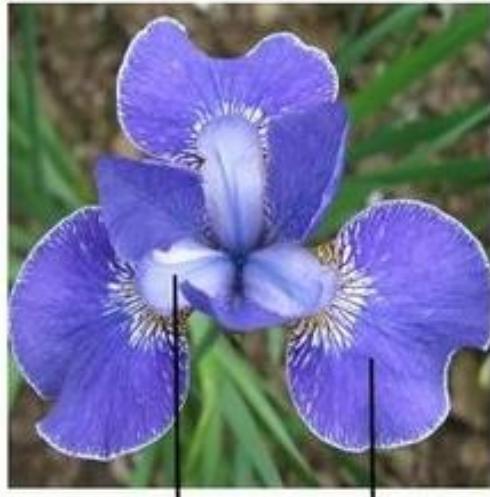
Praxisbeispiele

- Kundensegmentierung: Einteilung von Kunden anhand Kaufverhalten, Demografie oder Nutzungsverhalten
- Dokumenten- und Text-Cluster: Ähnliche Dokumente oder Textfragmente gruppieren (z.B. in der Themenanalyse)
- Bildanalyse: Auffindung ähnlicher Bildbereiche oder Objekterkennung anhand Merkmalen
- Weitere Anwendungsfälle: Bioinformatik (z.B. Genexpressionen), Anomalieerkennung, Empfehlungssysteme

Code Demo: Cluster Analyse des Iris Datensatz

demo_clusteranalyse.py

iris setosa



petal

sepal

iris versicolor



petal

sepal

iris virginica

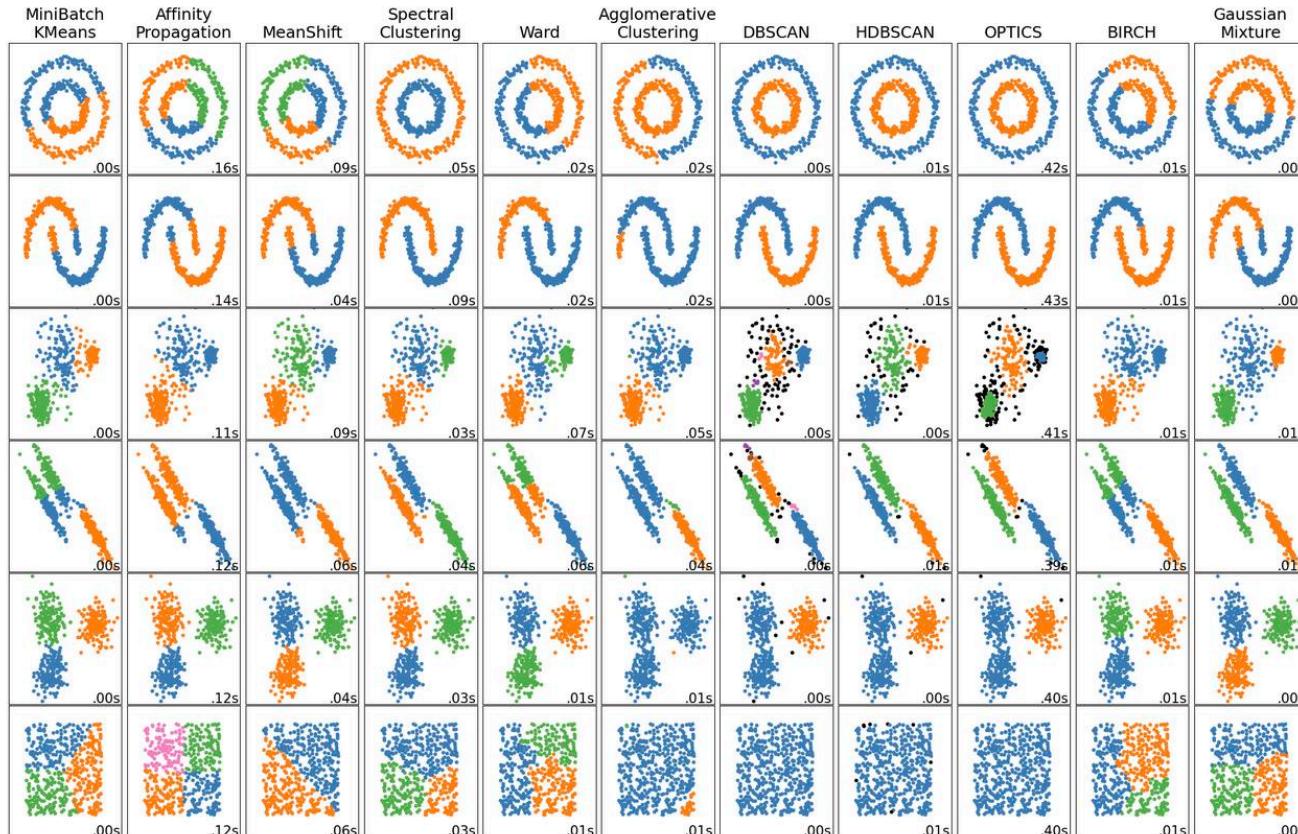


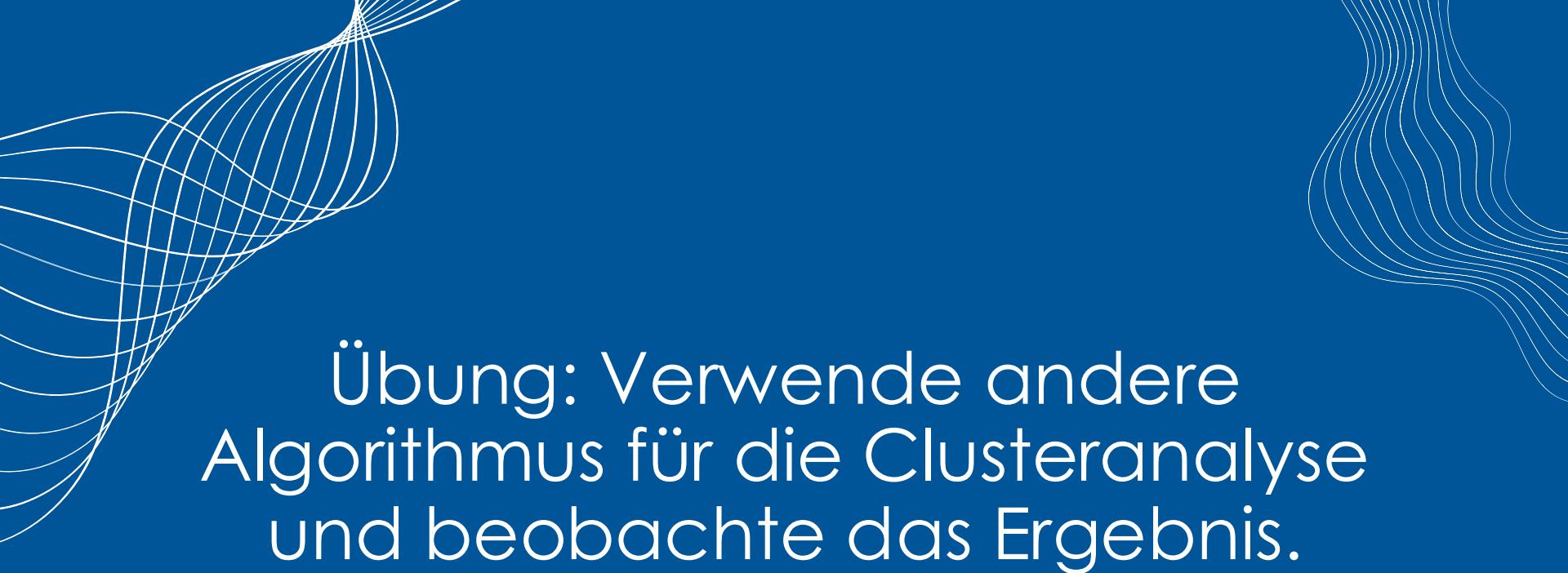
petal

sepal

<https://media2.dev.to/dynamic/image/width=1000,height=420,fit=cover,gravity=auto,format=auto/https%3A%2F%2Fdev-to-uploads.s3.amazonaws.com%2Fuploads%2Farticles%2F2tcrzvzwfn64ru9jb5p.png>

Weitere Clusteranalyse-Algorithmen





Übung: Verwende andere
Algorithmus für die Clusteranalyse
und beobachte das Ergebnis.

Übung: Erstelle eine Clusteranalyse für Kundendaten

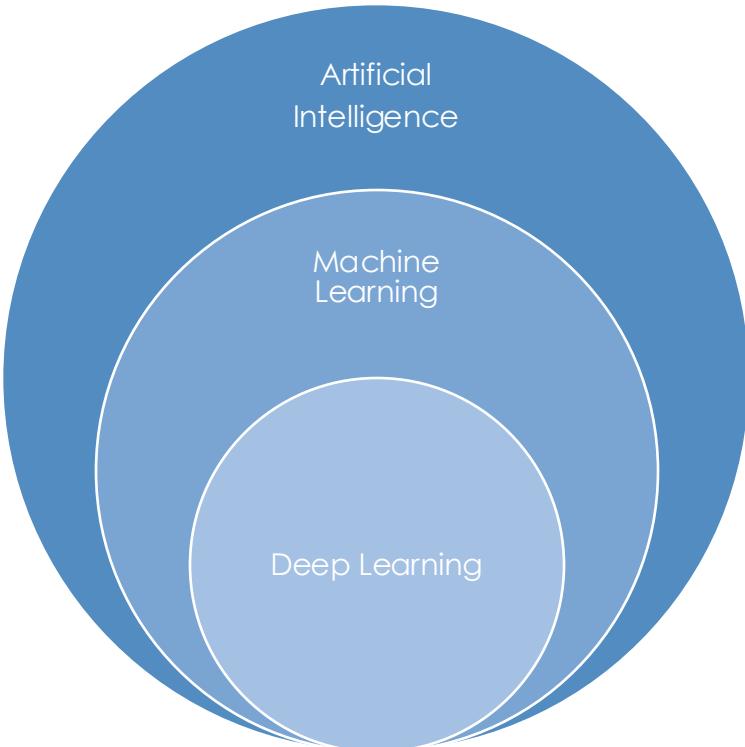
daten/Mall_Customers.csv

Zusammenfassung und Ausblick

- Kernidee: Datenpunkte in **sinnvolle Gruppen zusammenfassen**, ohne vorherige Labels
- Wichtigkeit k-Means: **Wahl der passenden Clusteranzahl** und geeigneter Merkmalsdarstellung zur Qualität der Ergebnisse
- Ausblick: **Weitere Verfahren** wie hierarchische oder dichte basierte Verfahren (z.B. DBSCAN) bieten unterschiedliche Vorteile
- Praxisempfehlung: **Datenaufbereitung und Visualisierung** (z.B. durch 2D-Projektionen) sind entscheidend für erfolgreiche Clusteranalyse

Inhaltlicher Ablauf

Tag	Thema
1	Einführung Pandas / Numpy
	Einführung in das Machine Learning
	Datenmanagement für Machine Learning
	Erste Prognose für den Machine Learning Task
	Vorstellung der Übungstasks
2	Modellevaluation
	Anwenden unterschiedlicher Modelle
	Ensemble Modellierung
	Anwenden unterschiedlicher Modelle auf Übungstasks
3	Automatisierte Parametersuche
	Regression
	Clusteranalyse
	Neuronale Netze
	Ausblick



Artificial Intelligence / Künstliche Intelligenz

Logiken und intelligentes Verhalten von Maschinen.
Keine klare Definition: Was ist intelligent?

Machine Learning

Aus einer Menge von Daten wird eine allg. Regel abgeleitet. Die Ableitung erfolgt automatisiert – d.h. durch einen Algorithmus und ohne explizite Anweisung durch den Programmierer.

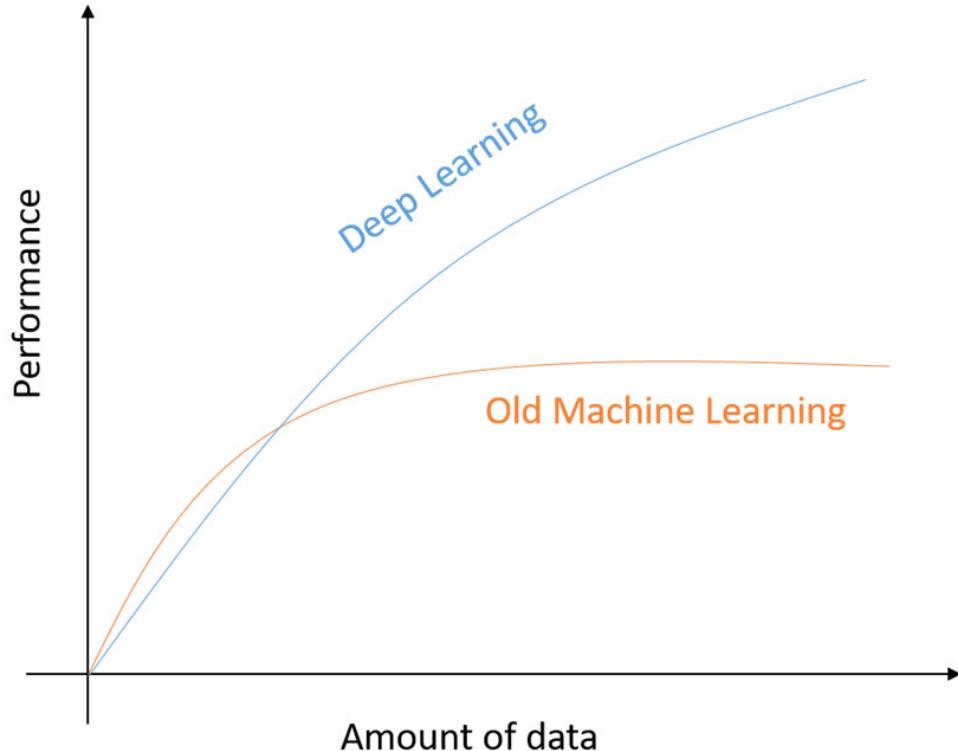
Deep Learning

Teilgebiet des Machine Learning. Meint die Anwendung Neuronaler Netze mit vielen verdeckten Schichten. Kann durch seine komplexe innere Struktur komplexe Zusammenhänge in Daten aufzeigen.

- Das aktuell erfolgreichste Modell im Machine Learning sind Neuronale Netze.
- Neuronale Netze können für alle vorgestellten Algorithmen Typen (Supervised, Unsupervised, Reinforcement) eingesetzt werden.
- Sie eignen sich insbesondere für Bild- und Sprachverarbeitung.
- Neuronale Netze sind in Schichten aufgebaut. Ab 2 Schichten spricht man von einem „tiefen“ Neuronalen Netz: **Deep Learning**.

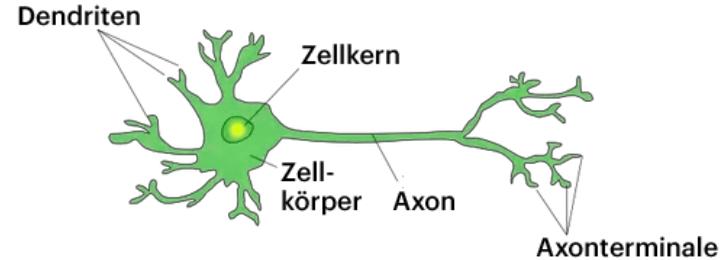
Eigenschaften Neuronaler Netze

- Feature Engineering ist einfacher
- Performance steigt mit Größe des Datensatzes und aktuell kein Ende in Sicht
- Training eines Modells ist aufgrund der Infrastruktur Anforderung teuer



Grundlagen biologischer Neuronen: Aufbau und Funktionsweise

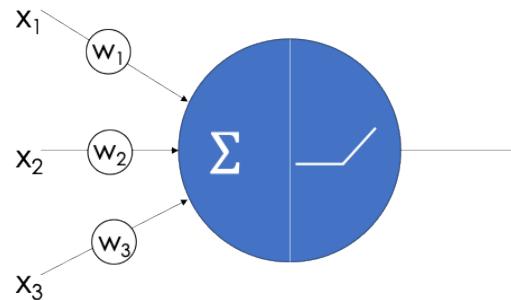
- Neuronale Grundstrukturen umfassen **Dendriten** (zur Aufnahme von Signalen), das **Soma** (Zellkörper) zur Verarbeitung der **eingehenden Informationen** und das **Axon** zur Weiterleitung elektrischer Impulse.
- Die **Informationsübertragung** erfolgt elektrisch entlang des Axons sowie chemisch über Neurotransmitter an den Synapsen.
- **Synapsen bilden Kontaktstellen** zwischen Neuronen und gewährleisten die chemische Signalweitergabe.
- Die Komplexität biologischer Neuronen dient als **Inspirationsquelle** für künstliche neuronale Netze.



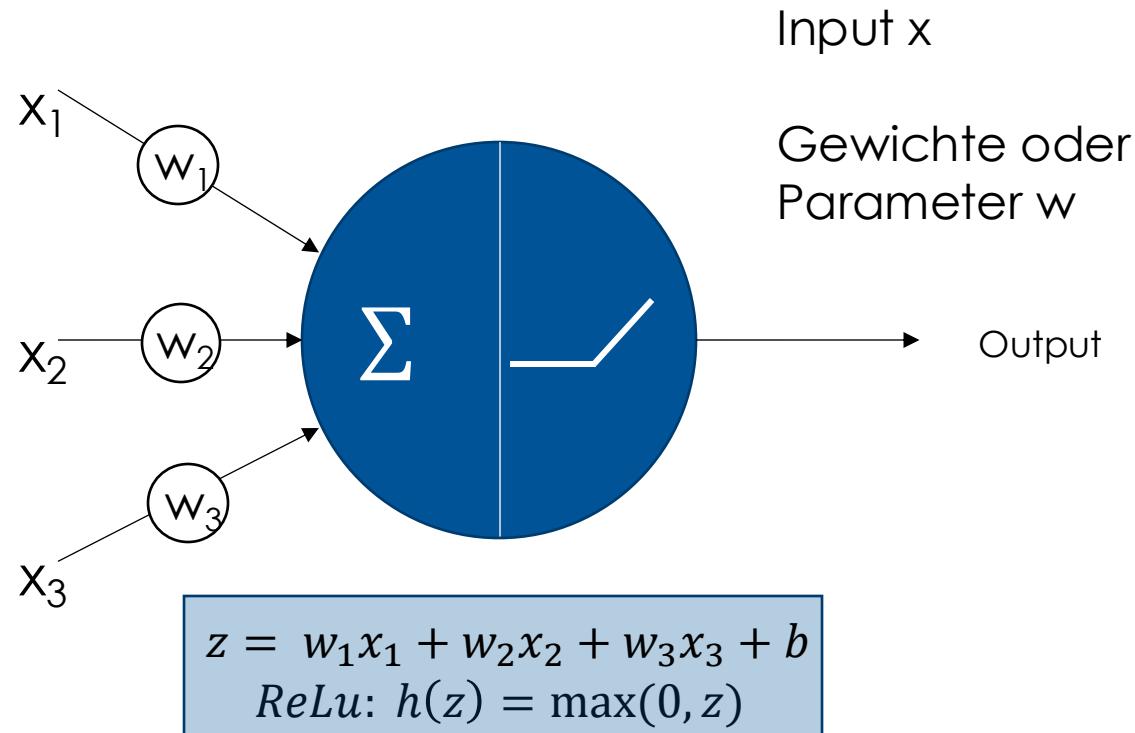
<https://10.wp.com/katzberger.ai/wp-content/uploads/2019/12/neuronen-netzwerk-mensch-maschine.png?fit=602%2C376&ssl=1>

Vergleich künstliches vs. biologisches Neuron: Ähnlichkeiten und Unterschiede

- Künstliche Neuronen sind **vereinfachte** mathematische **Modelle** biologischer Neuronen.
- Biologische Neuronen sind hochkomplex, weisen enorme Anpassungsfähigkeit auf und lernen kontinuierlich.
- In beiden Fällen werden Eingangssignale verarbeitet, um ein Ausgabe- bzw. Antwortsignal zu erzeugen.
- Während biologische Prozesse stark variieren und **chemisch-elektrisch** ablaufen, arbeiten künstliche Neuronen nach **deterministischen, reproduzierbaren** Berechnungen.



Neuronale Netze – das Neuron

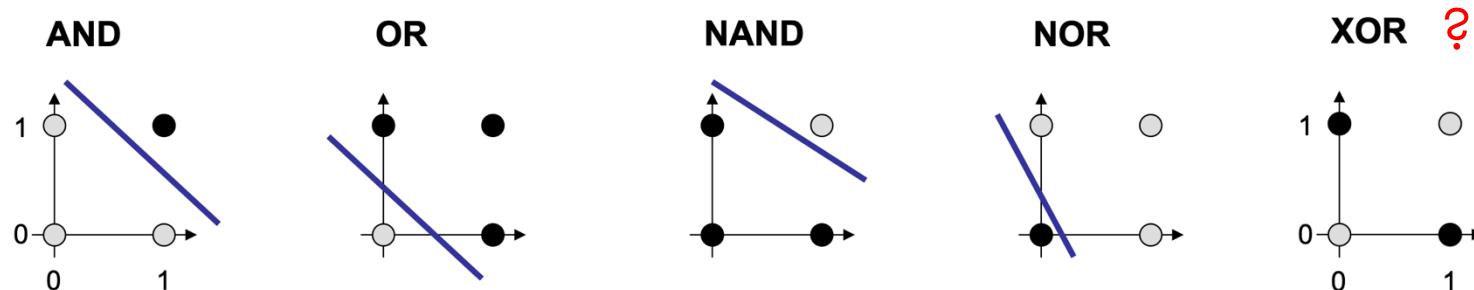


Code Demo: Einfaches Perceptron in NumPy implementieren

demo_perceptron.py

Historischer Überblick: Perceptron und frühe neuronale Netze

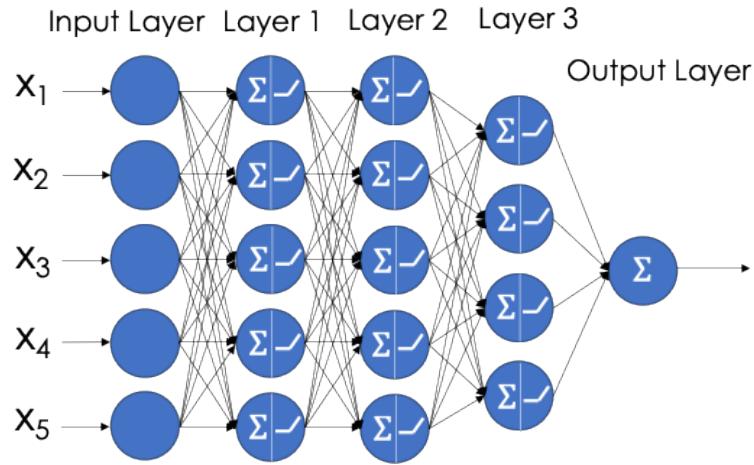
- Das **Perceptron** (1958) war ein einfaches, lineares Modell für **binäre Klassifikationsaufgaben**.
- Die **Unfähigkeit**, nichtlineare **Probleme wie XOR** zu lösen, führte zur Entwicklung von **Mehrschichtnetzwerken**.
- Der Durchbruch kam in den 1980er Jahren mit der **Backpropagation**, die tiefere Netze effektiv trainierbar machte.
- Diese frühen Ansätze legten die Grundlage für moderne Deep-Learning-Techniken.



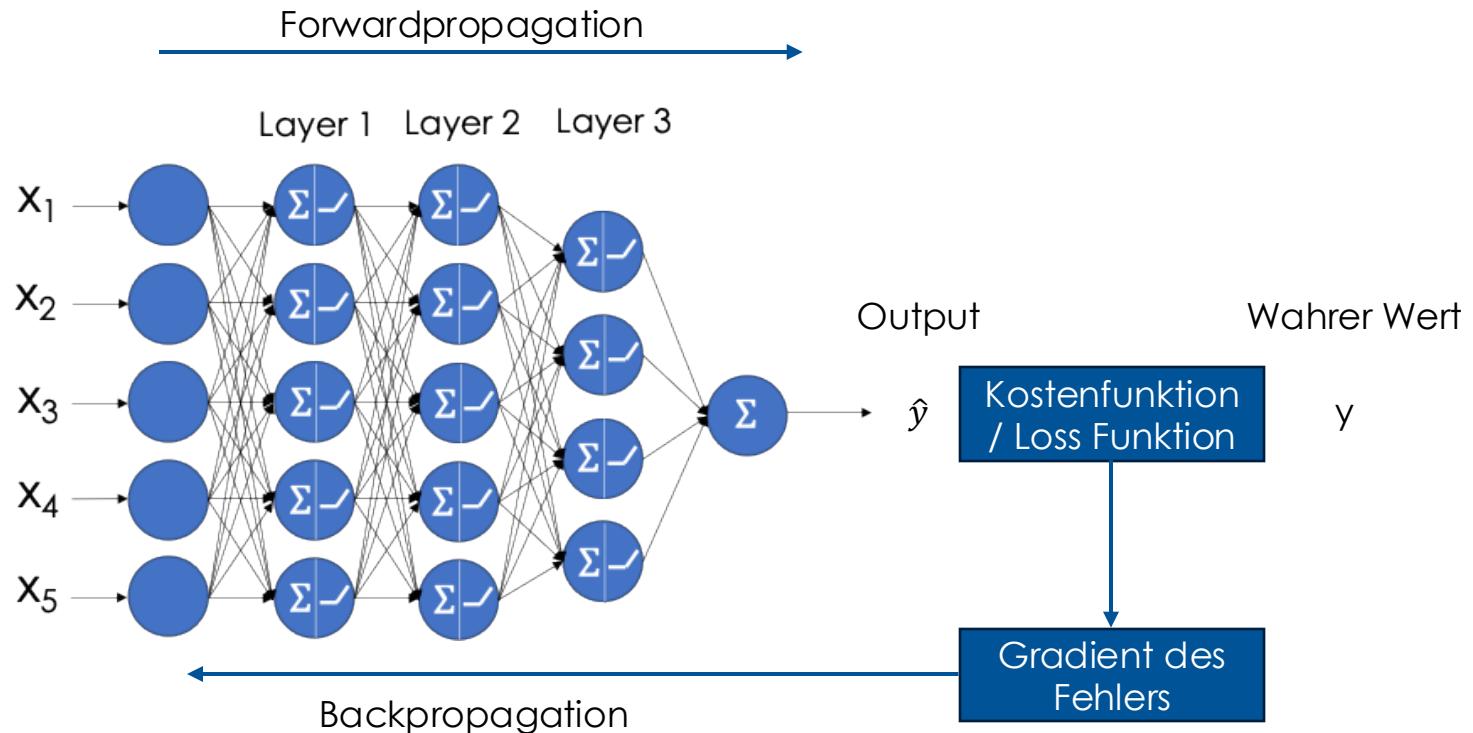
<https://ls11-www.cs.tu-dortmund.de/people/rudolph/teaching/lectures/POKS/WS2007-08/lecMeta.pdf>

Schichtenarchitektur: Input-, Hidden- und Output-Layer im Überblick

- Der **Input-Layer** nimmt die Rohdaten auf und gibt sie an die nächsten Schichten weiter.
- **Hidden-Layer** extrahieren schrittweise abstrakte Merkmale aus den Eingangsdaten.
- Der **Output-Layer** liefert letztlich die finale Vorhersage, beispielsweise eine Klasse oder einen numerischen Wert.
- Durch mehrere hintereinander geschaltete Schichten (Tiefe) können komplexe Muster erkannt werden.



Beispiel eines einfachen neuronalen Netzes: Datenfluss von Input bis Output

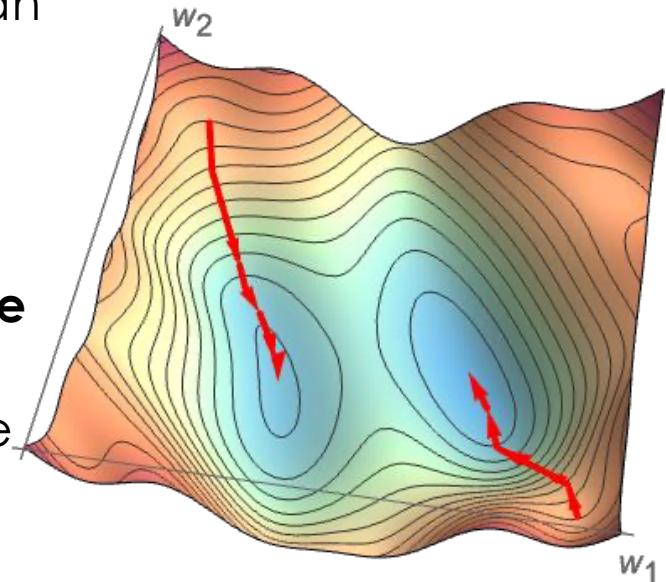


Beispiel eines einfachen neuronalen Netzes: Datenfluss von Input bis Output

- Die Eingabedaten passieren **Schicht für Schicht gewichtete Verbindungen**, die ihre Bedeutung anpassen.
- **Aktivierungsfunktionen** entscheiden, welche Signale weitergeleitet werden, um nichtlineare Beziehungen abzubilden.
- Über einen **Fehlervergleich (Loss)** zwischen Soll- und Ist-Ausgabe wird die Genauigkeit bestimmt.
- Durch **Anpassung der Gewichte anhand des Fehlers** lernt das Netz, seine Vorhersagen schrittweise zu verbessern.

Deep Learning – Wie werden die richtigen Gewichte gefunden?

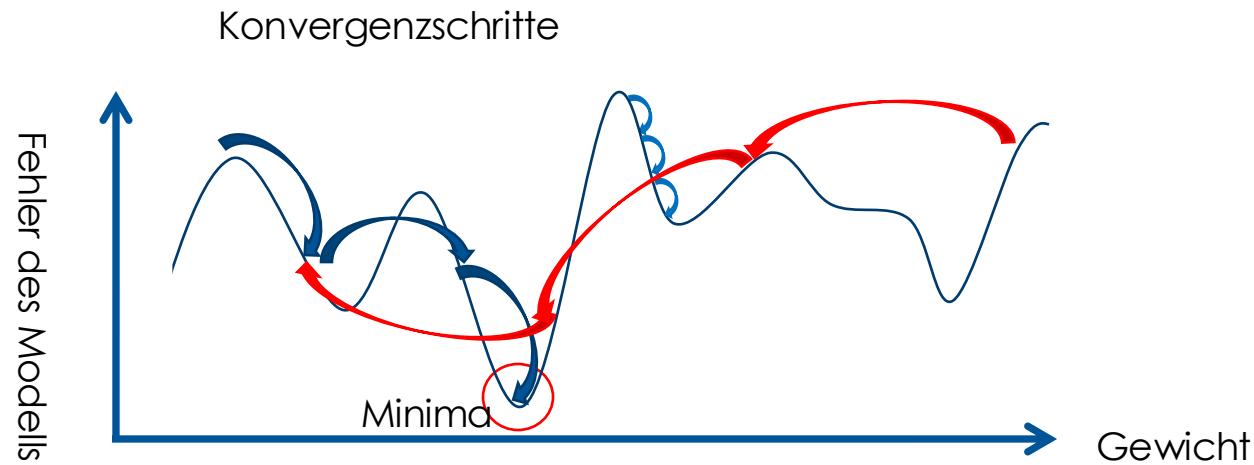
- Das Finden der richtigen Gewichte nennt man **Training**
- Das Training ist eine **iterative und ggf. lange Berechnung**
- Dabei werden die Daten dem Neuronalen Netzwerk **mehrfach gezeigt und die Gewichte Schrittweise angepasst**
- Der Algorithmus zur Anpassung der Gewichte heißt **Backpropagation**



<https://writings.stephenwolfram.com/2023/02/what-is-chatgpt-doing-and-why-does-it-work/>

Der Lernprozess

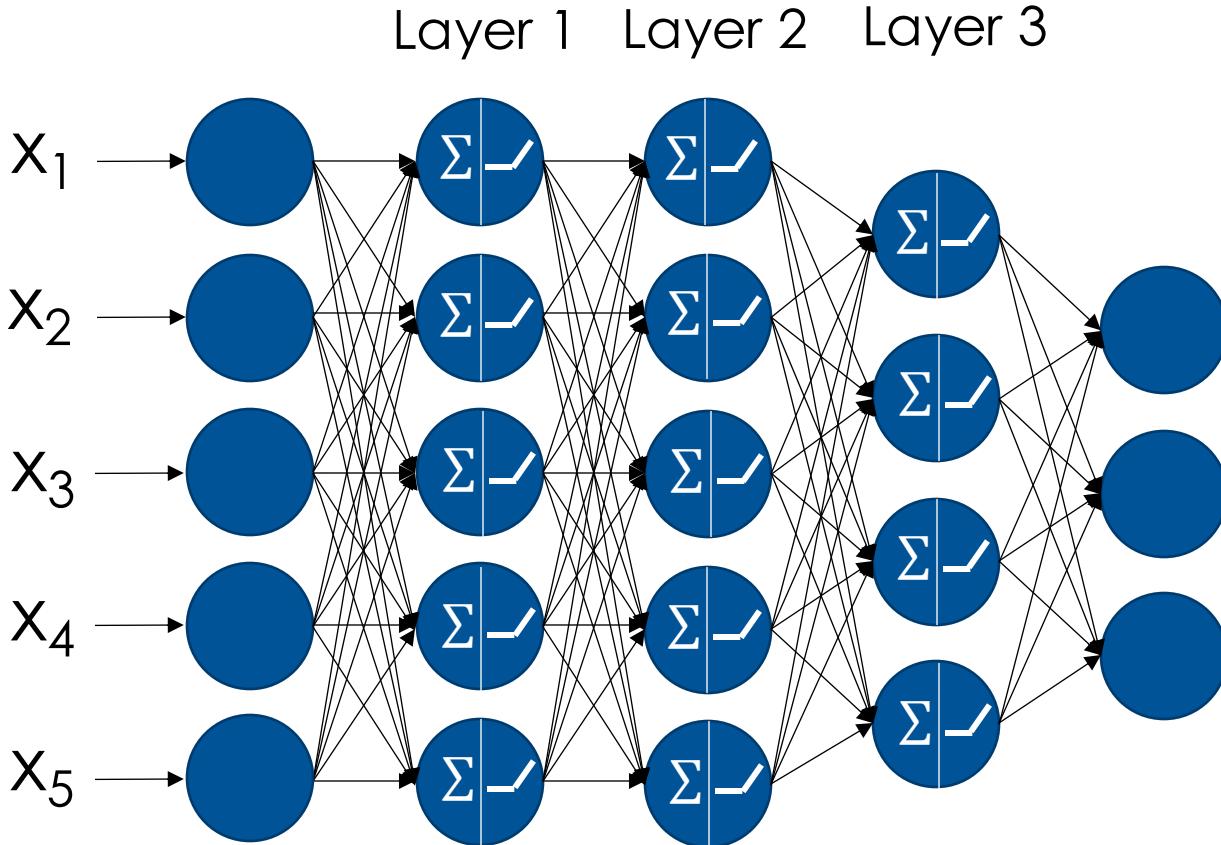
Eine zentrale Herausforderung bei der Modellierung von Neuronalen Netzen ist die Bestimmung der optimalen Lernrate. Problemstellung. Wird die Lernrate zu klein gewählt, neigt der Algorithmus dazu, in einem lokalen Minima zu konvergieren. Zu groß, konvergiert der Algorithmus nicht.



Begriffe

- = Neuronales Netz
- = Neural Net (NN)
- = Multilayer Perceptron (MLP)
- = (Mehrlagen Perzepron)
- = Feed Forward Network (nur Standard Layer)
(ab 2 Hidden Layers)
- = Tiefes Neuronales Netz
- = Deep Neural Net

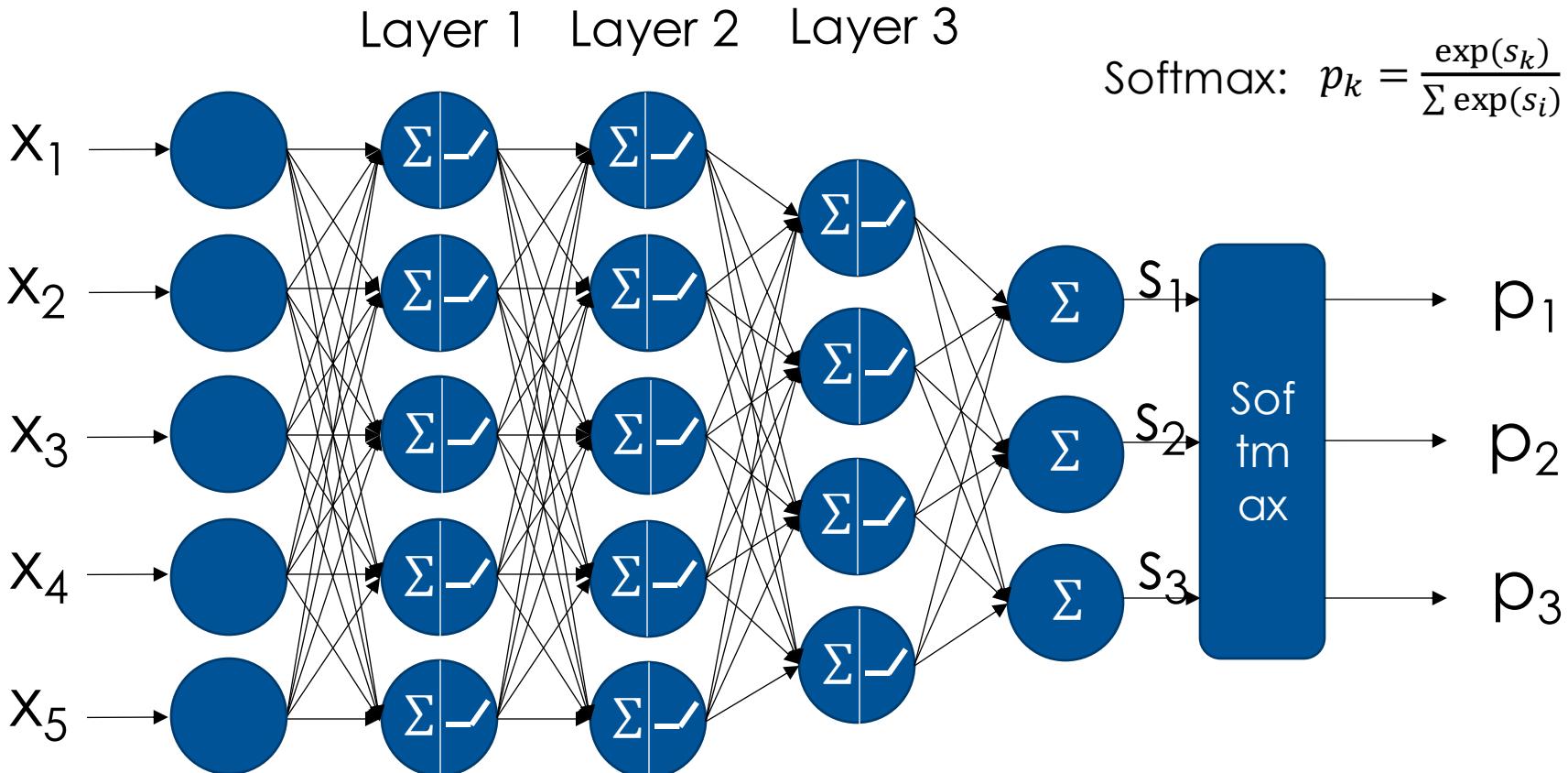
Neuronale Netze – Deep Learning



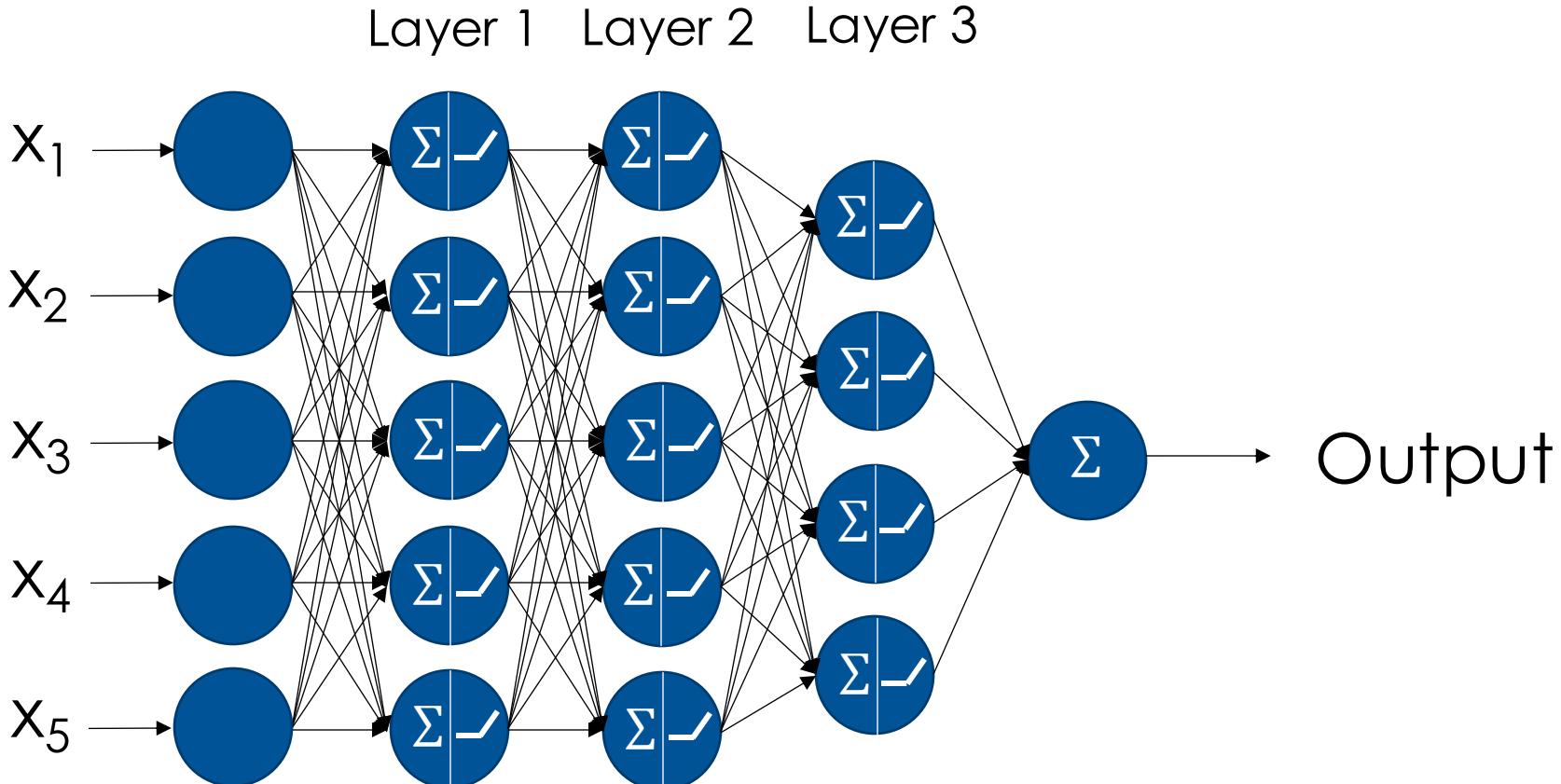
Fully-Connected: alle Neuronen in aufeinanderfolgenden Layers sind miteinander verbunden

Multi-Layer Perceptron (MLP): Fully-Connected Neuronales Netz mit >2 Layer.

Deep Learning - Klassifikation



Deep Learning - Regression



Playground

<https://playground.tensorflow.org/>

Übung

1. Verändere die Lernrate im Trainingsprozess. Starte mit der Lernrate "1", bis sich das Loss nicht mehr verändert. Wechsel dann auf 0.3 und trainiere weiter. Was beobachtest du? Kannst du damit Datensätze besser modellieren?
2. Nutze bei Klassifikation den letzten Datensatz (Spirale) und trainiere ein Neuronales Netz, das die Aufgabe löst. Welche Tricks helfen?
3. Erhöhe die Anzahl der Neuronen und Layer auf das Maximum. Wie verhält sich das Neuronale Netz?
4. Wie wirkt sich das „Noise“ auf das Loss aus? Warum?

Deep Learning – What else?

Größe des
Netzes

Hyperparameter

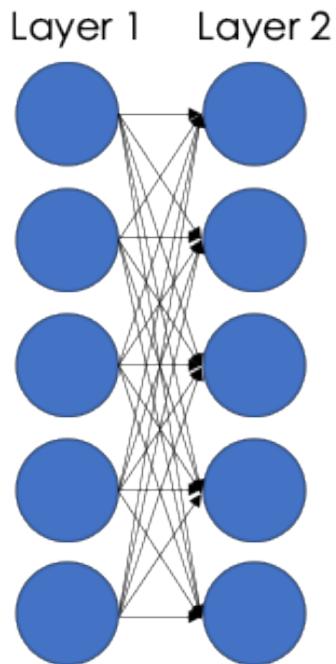
Jede Menge Tricks

Architektur

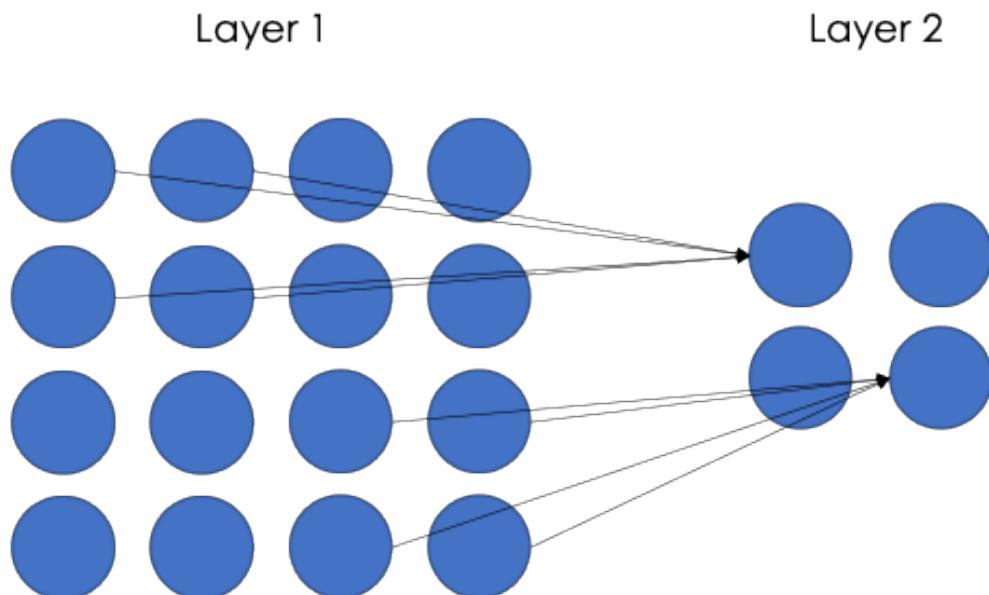
Neuronale Netze für Bildverarbeitung

Deep Learning – Convolutions

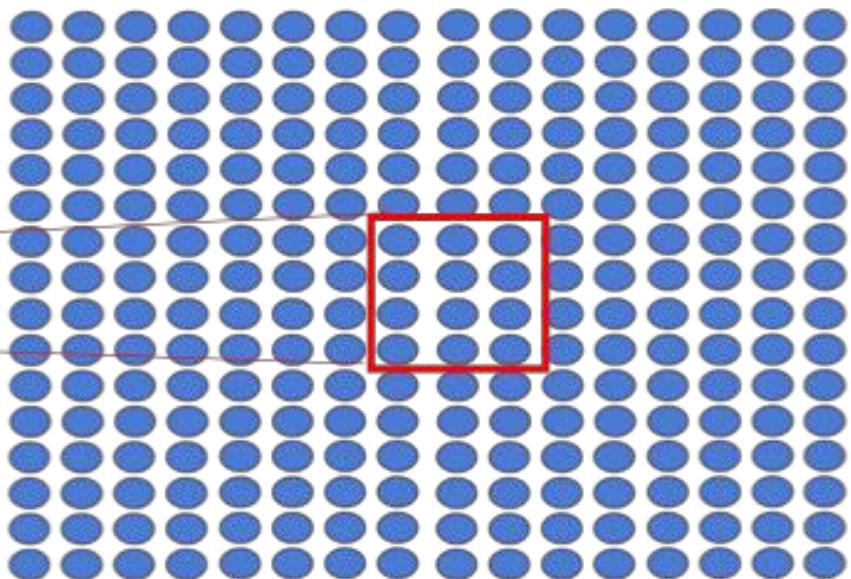
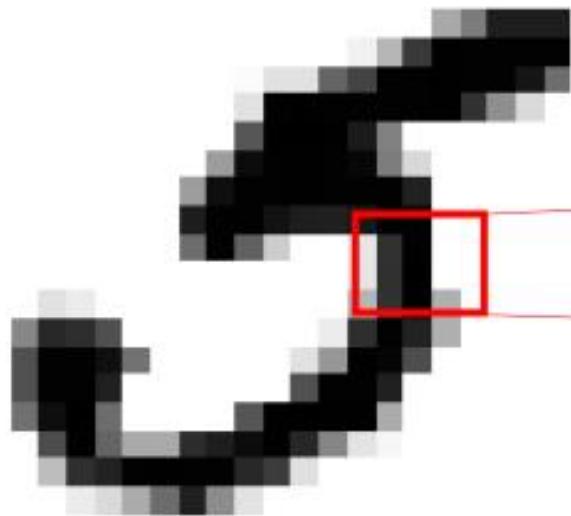
Bisher im Neuronalen Netz



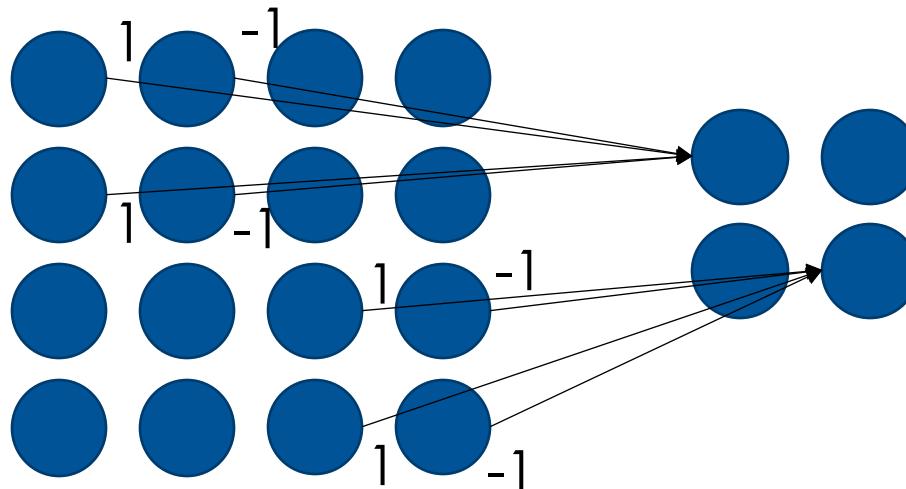
Im Convolutional Neuronal Net (CNN)



Deep Learning – Convolutions



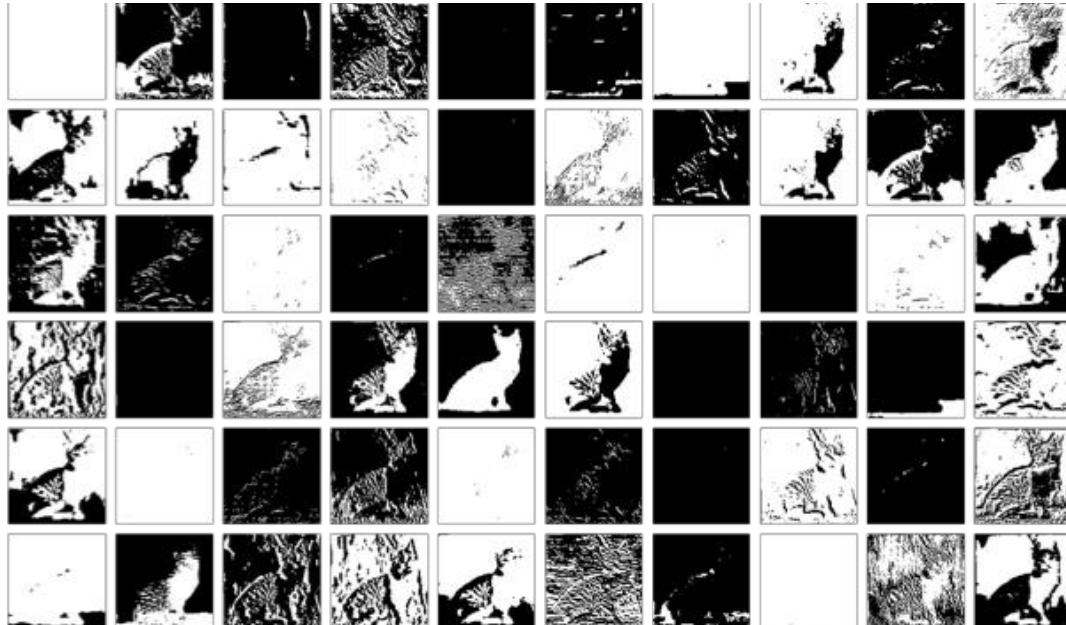
3. Convolutions als Filter



Convolutions können auch als Filter verstanden werden. In diesem Fall, um vertikale Kanten zu erkennen.



Deep Learning – Convolutions

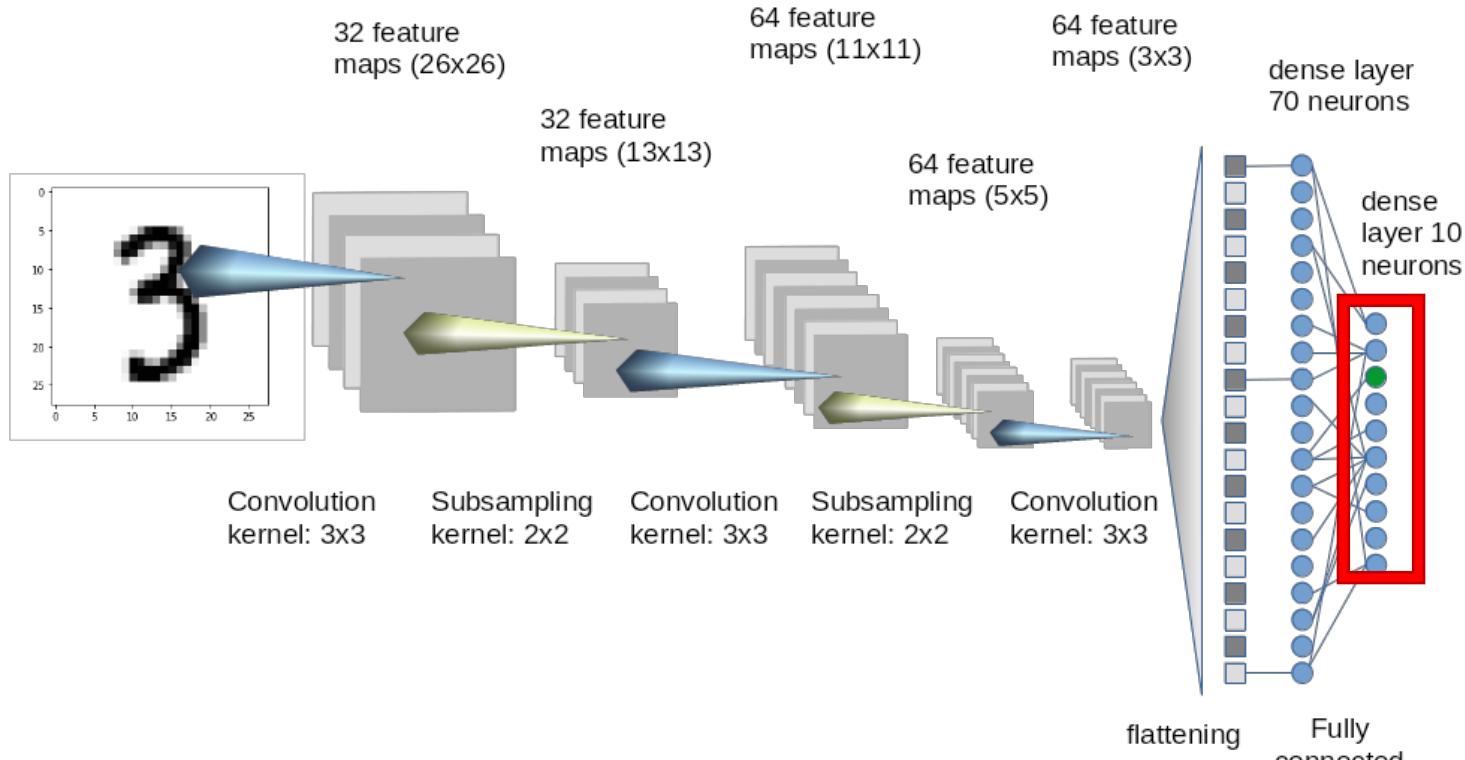


<https://writings.stephenwolfram.com/2023/02/what-is-chatgpt-doing-and-why-does-it-work/>

CNNs visualisiert

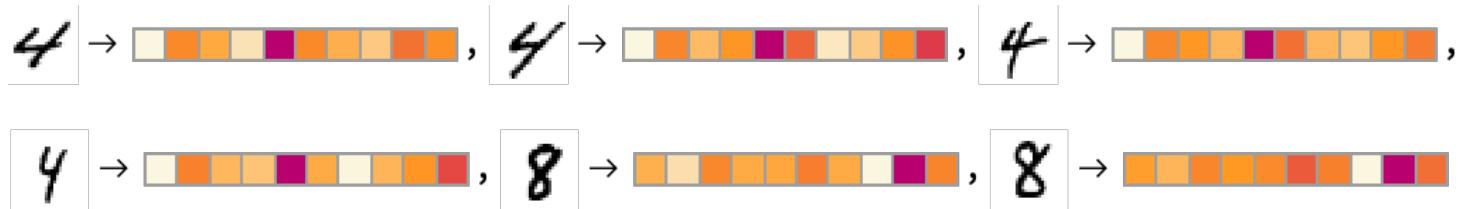
<https://poloclub.github.io/cnn-explainer/>

CNNs – Embeddings



CNNs – Embeddings

Embedding = Repräsentation (des Bildes) mit einem Vektor /
Array von Zahlen



Embeddings visualisiert

<https://projector.tensorflow.org>

Herausforderung

1. Das Training von CNNs ist aufwendig
2. Wenige Bilder vorhanden



Lösung

Ein CNN, das auf imagenet trainiert wurde, hat schon nützliche Convolutions gelernt. Nutze das CNN (und entferne gegebenenfalls die letzten Layers) und trainiere mit eigenen Daten weiter.

Übung

<https://teachablemachine.withgoogle.com/train>

1. Nutze 2 Klassen. Ab wie vielen Bildern wird die Klassifizierung gut?
2. Erweitere das Modell um eine dritte Klasse. Hat dies Auswirkungen auf die Performance?
3. Trainiere das Modell mit ungleicher Anzahl je Klasse. Stellst du einen Unterschied fest?



Neuronale Netze für Sprachverarbeitung

1. Von Wörtern zu Tokens

- Tokens können Wörter, Wortteile oder einzelne Zeichen sein

“Das ist ein Beispiel”  [“Das”, “ist”, “ein”, “Bei”, “spiel”]

Vorteile:

- Kleinere Vokabular-Größe, das auch seltene Wörter beinhalten (da diese in Sub-Wörter zerlegt werden)
- Auch Subfixe (z.B Beugungen von Verben) können erhalten bleiben

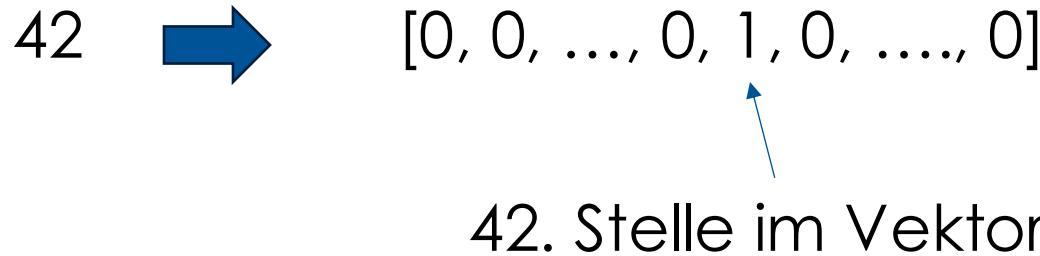
Deep Learning mit Text – Large Language Models

2. Von Tokens zu Encoding

- Jedes Token bekommt eine eigene ID

[“Das”, “ist”, “ein”, “Bei”, “spiel”] → [42, 13, 81, 32, 96]

- Anschließend Umwandlung in „One Hot“ Vektoren



Tokenizer in Action

<https://huggingface.co/spaces/Xenova/the-tokenizer-playground>

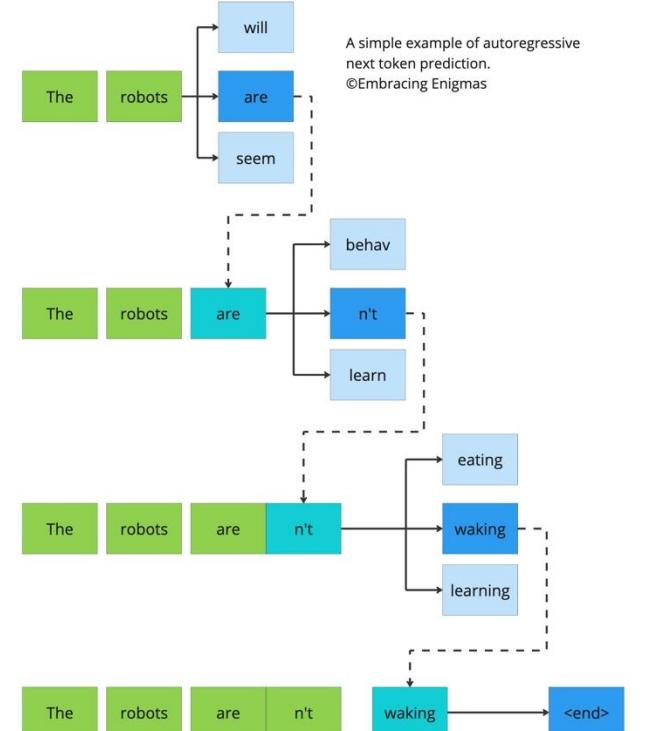
<https://gptforwork.com/tools/tokenizer>

LLMs zur Generierung von Text

- Auf Basis der bestehenden Token wird der **nächste Token vorhergesagt**
- Der vorhergesagte Token wird wieder in das Neuronale Netz „zurückgespeist“
- **Text-Generierung ist Klassifikation:** das Neuronale Netz weist den Tokens Wahrscheinlichkeiten zu. Im letzten Layer ist für jeden Token 1 Neuron mit entsprechendem Score.
- Aus Text kann so ein **supervised Training** erstellt werden

Next-Token Prediction: Grundlagen der Textgenerierung

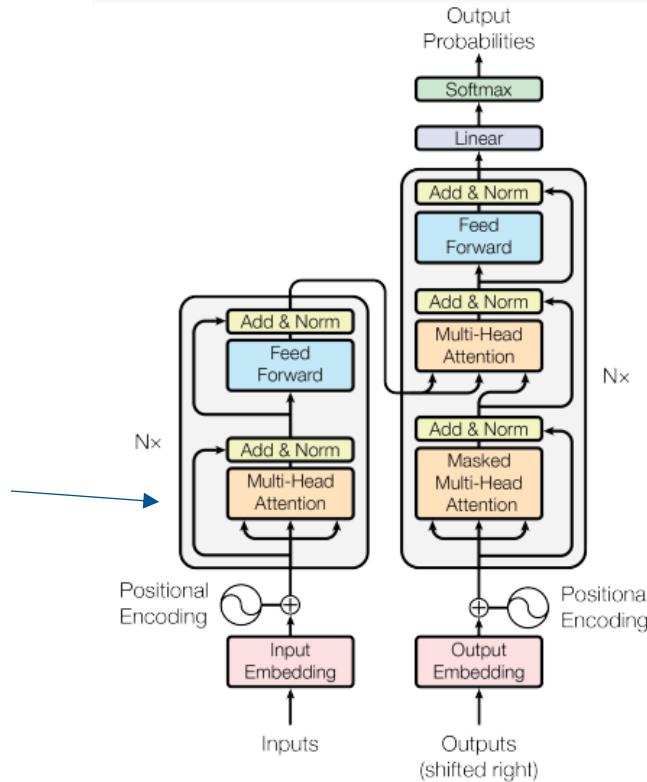
- Beim Next-Token-Prediction-Training sagt das Modell das **nächste Wort auf Basis des bisherigen Kontextes** vorher.
- Dieses unüberwachte Trainingsverfahren nutzt riesige Textkorpora, um statistische Sprachmuster zu erlernen.
- Das Modell entwickelt ein **implizites Verständnis für Grammatik, Syntax und semantische Muster**.
- Next-Token Prediction bildet das **Fundament für generative Sprachmodelle** wie GPT.



https://substackcdn.com/image/fetch/f_auto,q_auto,g_auto,g_fi,progressive/steepep/https%3A%2F%2Fsubstack-post-pending.s3.amazonaws.com%2Fpublic%2Fimages%2F943efc65-c436-4914-bd9-13c398c6dc5_1876x2109.jpeg

LLMs - Transformer

Attention is all
you need



Attention Mechanismus

<https://poloclub.github.io/transformer-explainer/>

LLMs zum Erzeugen von Embeddings

- Nutze das LLM, um **Embeddings von Text** zu erzeugen.
Speichere diese Embeddings in einer Datenbank ab.
- Diese Datenbank ist dann für LLMs zugänglich. Der **Input Prompt wird dann nach Ähnlichkeit** (der Input Embeddings zu den Datenbank Embeddings) durchsucht.
- Die Texte mit der **größten Ähnlichkeit** werden im Context des Prompts eingebunden.

RAGs – Retrieval Augmented Generation

- Der Prompt wird **bereichert um den Kontext**, der in der Datenbank gefunden wurde

Beispiel Anwendungen

- Customer Support Chatbots
- Suchmaschinen
- Code Assistant bei der Software Entwicklung
- Unternehmens Knowledge-Base per Chat zur Verfügung stellen

Perplexity, GPTs, Notebook LM

<https://www.perplexity.ai>

<https://chatgpt.com>

<https://notebooklm.google.com>

- Szenario Chatbot - LLM um Wissen oder Fachsprache erweitern
 - Trainiere ein bestehendes Modell mit neuen Daten weiter, um eigenes Wissen oder Fachsprache in das LLM zu integrieren.
- Szenario LLM für andere Aufgabe nutzen
 - Nutze bestehendes LLM für anderen Zweck, z.B. Klassifizierung von Text. Dafür den letzten Layer des Netzwerkes abschneiden und die Anzahl der Neuronen anpassen. Anschließend mit Daten trainieren.

LLMs mit eigenen Daten schlau machen

Prompt Engineering / In Context Learning



Sehr simpel und schnell



Begrenzt durch Context Window.
Beschränkt in der Lernfähigkeit.

Embeddings in Datenbank / RAG



Schnell und einfach erweiterbar und updatebar mit neuem Wissen.



Ggf. komplex, da Datenbank benötigt wird



Stil und Fachsprache kann gelernt werden.
Wissen wird am besten übernommen.
Langfristig Kosten sparen.



Kosten für Training
Ggf. Komplexität im Training
Anforderung an Datengröße

LLMs – für die Software Entwicklung

- Code Completion (z.B. Copilot)
 - Ergänzung von Code Zeilen
 - Schreiben von Kommentaren und Code schreiben lassen
- Prompt: „Schreibe Unit Tests für folgende Funktion ...“
- Prompt: „Schreibe ein Python Skript, das eine Visualisierung erstellt...“
- Architektur Fragestellungen bzw. Brainstorming im Chat
- Fehler Log Analysieren und Lösungen vorschlagen
- Schreibe eine Dokumentation für den Code
- Erkläre mir, was der folgende Code macht
- Agenten für Deployment (z.B. Replit)
- IDEs mit integrierter KI bieten zusätzliche Funktionalität (z.B. Cursor AI)
 - Indexierung von Dokumentationen
 - Composer: Schreiben und Editieren von mehreren Files nach Anweisung im Prompt



- Idee: LLMs können auch strukturierten Output generieren, z.B. im JSON-Format.
Dies kann wiederum zum Aufrufen einer API genutzt werden.
 - Web Suchanfragen ausführen
 - Taschenrechner aufrufen
 - Daten von externen Quellen einbinden
 - Termine erstellen
 - ... alles, was eine API hat

Large Language Models - Temperatur

Das Beste an KI ist die Fähigkeit zu ...

Wort	Wahrscheinlichkeit
lernen	5.4 %
verstehen	4.1 %
prognostizieren	3.6 %
entdecken	1.1 %
...	...

- Die Wahl des wahrscheinlichsten Wortes (Temperatur =0) erzeugt oft schlechte Text
- mit einer hohen Temperatur (> 1) werden auch unwahrscheinliche Worte gewählt
- Damit lässt sich die „Kreativität“ des Modells steuern
- In der Praxis wird oft eine Temperatur von 0.7 gewählt

Large Language Models – Context Length

- Context Length (oder auch Window) ist eine Beschränkung der Anzahl an Tokens, die der Transformer verarbeiten kann
- Die Context Length inkludiert den Input, den Output und gegebenenfalls den vorigen Chatverlauf
- Aktuell GPT-4: 128000 Tokens, Gemini: 2 Millionen Tokens
- Zum Vergleich: die Bibel hat ~1.4 Millionen Token

Gewichte im LLM

Llama 3.1: 405 Milliarden

PaLM 2: 340 Milliarden

GPT-4: ~1 Billionen (geschätzt)

Zum Vergleich: im Gehirn

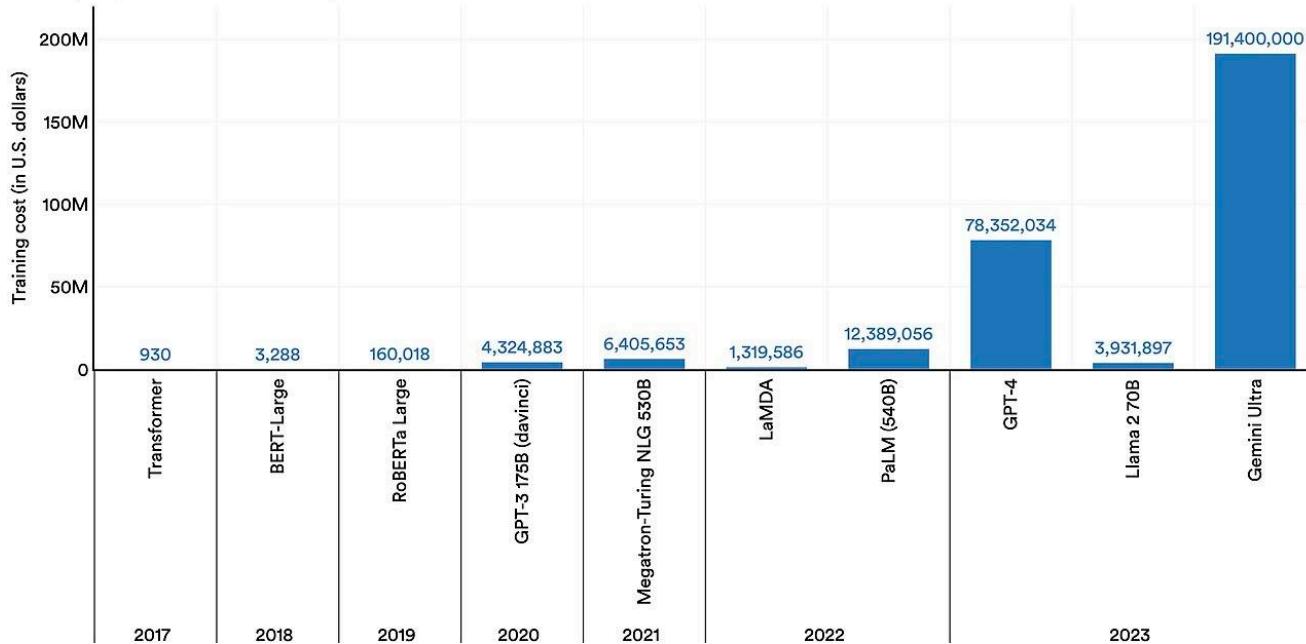
100 Milliarden – 1 Billionen
Nervenzellen

> 100 Billionen Synapsen
(Verbindungen)

LLMs – Training

Estimated training cost of select AI models, 2017–23

Source: Epoch, 2023 | Chart: 2024 AI Index report



https://en.wikipedia.org/wiki/Large_language_model

Datacenter xAI

100.000 Nvidia H100

Stromverbrauch pro H100

~1 Haushalt

Kosten pro H100

~40.000 €



Nvidia H100



Tokens

100 Milliarden – 20
Billionen

Speicherplatz

Einstellig Terabyte

Für GPT-3

Dataset	Anzahl Tokens	Anteil am Training
Common Crawl	410 Milliarden	60%
WebText2	19 Milliarden	22%
Books1	12 Milliarden	8%
Books2	55 Milliarden	8%
Wikipedia	3 Milliarden	3%

Welches ist aktuell das beste LLM?

Rank* (UB)	Rank (StyleCtrl)	Model	Arena Score	95% CI	Votes	Organizatio	License	Knowledge Cutoff
1	3	Gemini-2.0-Flash-Thinking-Exp-01-21	1382	+8/-6	6437	Google	Proprietary	Unknown
1	1	Gemini-Exp-1206	1374	+5/-4	22116	Google	Proprietary	Unknown
2	8	Gemini-Exp-1121	1365	+5/-4	17338	Google	Proprietary	Unknown
3	1	ChatGPT-4o-latest_(2024-11-20)	1365	+4/-4	35328	OpenAI	Proprietary	Unknown
3	4	Gemini-2.0-Flash-Thinking-Exp-1219	1363	+5/-5	17083	Google	Proprietary	Unknown
3	1	DeepSeek-R1	1357	+12/-13	1883	DeepSeek	MIT	Unknown
5	5	Gemini-2.0-Flash-Exp	1356	+4/-4	20939	Google	Proprietary	Unknown
6	1	o1-2024-12-17	1352	+6/-6	9230	OpenAI	Proprietary	Unknown
6	11	Gemini-Exp-1114	1347	+6/-5	17095	Google	Proprietary	Unknown
10	4	o1-preview	1335	+3/-3	33186	OpenAI	Proprietary	2023/10
11	11	DeepSeek-V3	1317	+6/-5	13640	DeepSeek	DeepSeek	Unknown
11	15	Step-2-16K-Exp	1305	+9/-7	4533	StepFun	Proprietary	Unknown

- Neuronale Netze funktionieren besser, wenn die Daten Skaliert sind.
- Hierfür eignet sich der Standardscaler.
- Mit der Methode `.inverse_transform` können die Daten wieder in die ursprüngliche Range zurück transformiert werden.

Abschluss Übungstask – Nutzung von Neuronalen Netzen

Nutze Neuronale Netze für Wetter Klassifikation und Regression.
Werden die Ergebnisse besser?

Inhaltlicher Ablauf

Tag	Thema
1	Einführung Pandas / Numpy
	Einführung in das Machine Learning
	Datenmanagement für Machine Learning
	Erste Prognose für den Machine Learning Task
	Vorstellung der Übungstasks
2	Modellevaluation
	Anwenden unterschiedlicher Modelle
	Ensemble Modellierung
	Anwenden unterschiedlicher Modelle auf Übungstasks
3	Automatisierte Parametersuche
	Regression
	Clusteranalyse
	Neuronale Netze
	Ausblick

Ausblick – wie weiter lernen?

- Kaggle: <https://www.kaggle.com>

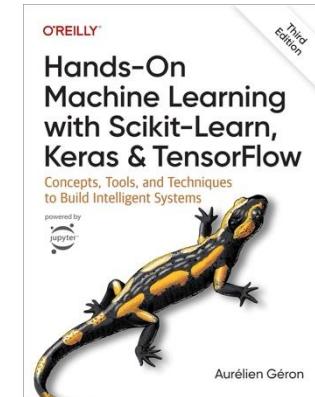
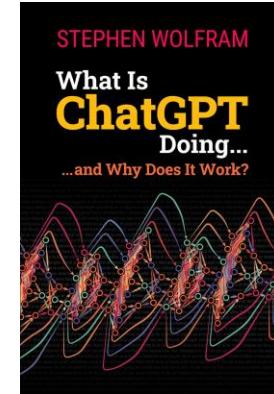
kaggle

- Datensätze downloaden und analysieren – Übung macht den Meister: <https://archive.ics.uci.edu>



Ausblick - Bücher

- Stephen Wolfram – What Is ChatGPT Doing ... and Why Does It Work:
<https://writings.stephenwolfram.com/2023/02/what-is-chatgpt-doing-and-why-does-it-work/>
- Aurelien Geron – Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems



Glossar

Glossar – Teil 1

Aktivierungsfunktion: Bestimmt, ob ein Neuron aktiviert ist und ein Signal aussendet.

AUC (Area Under the Curve): Metrik zur Bewertung der Leistung eines Klassifikators anhand der Fläche unter der ROC-Kurve.

Backpropagation: Algorithmus, der die Gewichte der Neuronen in Abhängigkeit von dem erzeugten Fehler des Gesamtmodells anpasst.

Bagging: Ensemble-Methode, bei der mehrere Modelle parallel trainiert werden und ihre Ergebnisse kombiniert werden, um die Vorhersagegenauigkeit zu verbessern. Ein bekanntes Beispiel ist der Random Forest-Algorithmus.

Boosting: Ensemble-Methode, bei der Modelle sequenziell trainiert werden, wobei jedes Modell versucht, die Fehler des vorherigen Modells zu korrigieren. Beispiele sind Adaptive Boosting und Gradient Boosting.

Deep Learning: Teilgebiet des Machine Learning, das sich auf die Anwendung neuronaler Netze mit vielen verdeckten Schichten ("Hidden Layers") konzentriert. Ermöglicht die Darstellung komplexer Zusammenhänge in Daten.

Entscheidungsbaum: Modell, das eine baumartige Struktur verwendet, um Entscheidungen zu treffen, indem es Daten anhand von Entscheidungsregeln in verschiedene Kategorien aufteilt.

Feature: Input-Variable, die zur Vorhersage der Zielgröße verwendet wird.

Gradient Boosting: Ensemble-Methode, die Entscheidungsbäume sequentiell trainiert, wobei jeder Baum die Fehler des vorherigen Baums korrigiert.

Grid-Search: Verfahren zur Optimierung der Hyperparameter eines Modells durch systematisches Ausprobieren verschiedener Parameterkombinationen.

Glossar – Teil 2

Klassifikation: Modell, das Daten in diskrete Kategorien einteilt, z. B. "Regentag" oder "kein Regentag".

Kreuzvalidierung: Verfahren zur Evaluation der Vorhersagegenauigkeit eines Modells durch Aufteilung des Datensatzes in mehrere Teilmengen (z.B. k-Fold-Kreuzvalidierung).

Lernrate: Parameter, der steuert, wie stark die Gewichte des Neuronalen Netzes bei jeder Iteration angepasst werden.

Machine Learning: Entwicklung von Algorithmen und Modellen, die selbstständig Muster in Daten finden und daraus lernen. Es wird eine allgemeine Regel aus den Daten abgeleitet, ohne explizite Anweisung durch den Programmierer.

Modell: Stellt den Zusammenhang zwischen Features und der Zielgröße her.

Modell-Evaluation: Verfahren, um die Vorhersagegenauigkeit eines Modells zu bestimmen.

Neuronale Netze: Bestehen aus miteinander verbundenen Knoten ("Neuronen") in Schichten, die Eingabedaten verarbeiten und Informationen weitergeben. Sie zeichnen sich durch ihre Fähigkeit aus, komplexe, nichtlineare Beziehungen zu erkennen.

Overfitting: Das Modell lernt die Trainingsdaten "auswendig" und kann neue Daten nicht gut vorhersagen.

Propagierungsfunktion: Funktion, die innerhalb eines Neurons alle Eingabesignale verdichtet.

Glossar - Teil 3

Random Forest: Ensemble-Methode, die mehrere Entscheidungsbäume parallel trainiert und deren Vorhersagen kombiniert.

Regression: Modell, das einen kontinuierlichen Wert vorhersagt, z. B. die Temperatur.

Reinforcement Learning: "Bestärkendes Lernen" - Algorithmen lernen durch Interaktion mit einer Umgebung und erhalten Belohnungen für erfolgreiche Aktionen.

ReLU (Rectified Linear Unit): Aktivierungsfunktion, die für positive Eingabewerte den Wert selbst zurückgibt und für negative Eingabewerte Null

ROC-Kurve (Receiver Operating Characteristic Curve): Grafische Darstellung der Sensitivität und Spezifität eines Klassifikators bei verschiedenen Schwellenwerten.

Supervised Learning: "Überwachtes Lernen" - Der Algorithmus lernt aus Daten, bei denen das korrekte Ergebnis bereits bekannt ist.

Testdaten: Datensatz, der verwendet wird, um die Performance des Modells auf unbekannten Daten zu evaluieren.

Trainingsdaten: Datensatz, der zum Trainieren des Modells verwendet wird.

Unsupervised Learning: "Unüberwachtes Lernen" - Der Algorithmus lernt aus Daten, bei denen kein vorgegebenes Ergebnis existiert, z.B. bei der Clusteranalyse oder Anomalieerkennung.

Vanishing Gradient Problem: Problem in Neuronalen Netzen, bei dem die Gradienten bei der Backpropagation in den frühen Schichten immer kleiner werden, was das Lernen des Modells behindert.

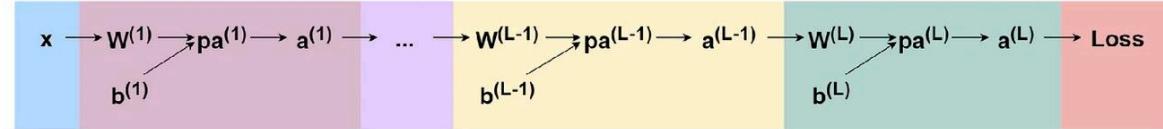
XGBoost (Extreme Gradient Boosting): Erweiterung von Gradient Boosting mit zusätzlichen Features zur Verbesserung der Leistung und Geschwindigkeit.

Backup

Backpropagation

Ziel: Gewichte anpassen

$$W_{neu}^{(L)} = W_{alt}^{(L)} - \eta \frac{\partial Loss}{\partial W^{(L)}}$$



Wie berechnet sich $\frac{\partial Loss}{\partial W^{(L)}}$: ?

$$Loss := \frac{1}{2}(\mathbf{a}^{(L)} - \mathbf{y})^2$$

Dafür muss zunächst die Ableitung des Loss in Richtung $a^{(L)}$ berechnet werden:

$$\frac{\partial Loss}{\partial \mathbf{a}^{(L)}} = \frac{\partial \frac{1}{2}(\mathbf{a}^{(L)} - \mathbf{y})^2}{\partial \mathbf{a}^{(L)}} = \mathbf{a}^{(L)} - \mathbf{y}$$

Im nächsten Schritt wird die Ableitung des Loss nach $p\mathbf{a}^{(L)}$ berechnet:

$$\frac{\partial Loss}{\partial p\mathbf{a}^{(L)}} = \frac{\partial Loss}{\partial \mathbf{a}^{(L)}} \frac{\partial \mathbf{a}^{(L)}}{\partial p\mathbf{a}^{(L)}} = \frac{\partial Loss}{\partial \mathbf{a}^{(L)}} \sigma'(\mathbf{p}\mathbf{a}^{(L)})$$

Nun kann das Loss in Richtung $W^{(L)}$ berechnet werden:

$$\frac{\partial Loss}{\partial \mathbf{W}^{(L)}} = \frac{\partial Loss}{\partial p\mathbf{a}^{(L)}} \frac{\partial p\mathbf{a}^{(L)}}{\partial \mathbf{W}^{(L)}}$$

$$\frac{\partial p\mathbf{a}^{(L)}}{\partial \mathbf{W}^{(L)}} = \frac{\partial (\mathbf{b}^{(L)} + \mathbf{W}^{(L)} \mathbf{a}^{(L-1)})}{\partial \mathbf{W}^{(L)}} = \mathbf{a}^{(L-1)}$$