HSBI

Hochschule
Bielefeld

University of
Applied Sciences
and Arts

Studiengang Informatik Fachbereich Campus Minden

Hochschule Bielefeld
University of Applied Sciences and Arts

Datenanalyse und Einführung in Maschinelles Lernen WS 2025/26

Maschinelles Lernen

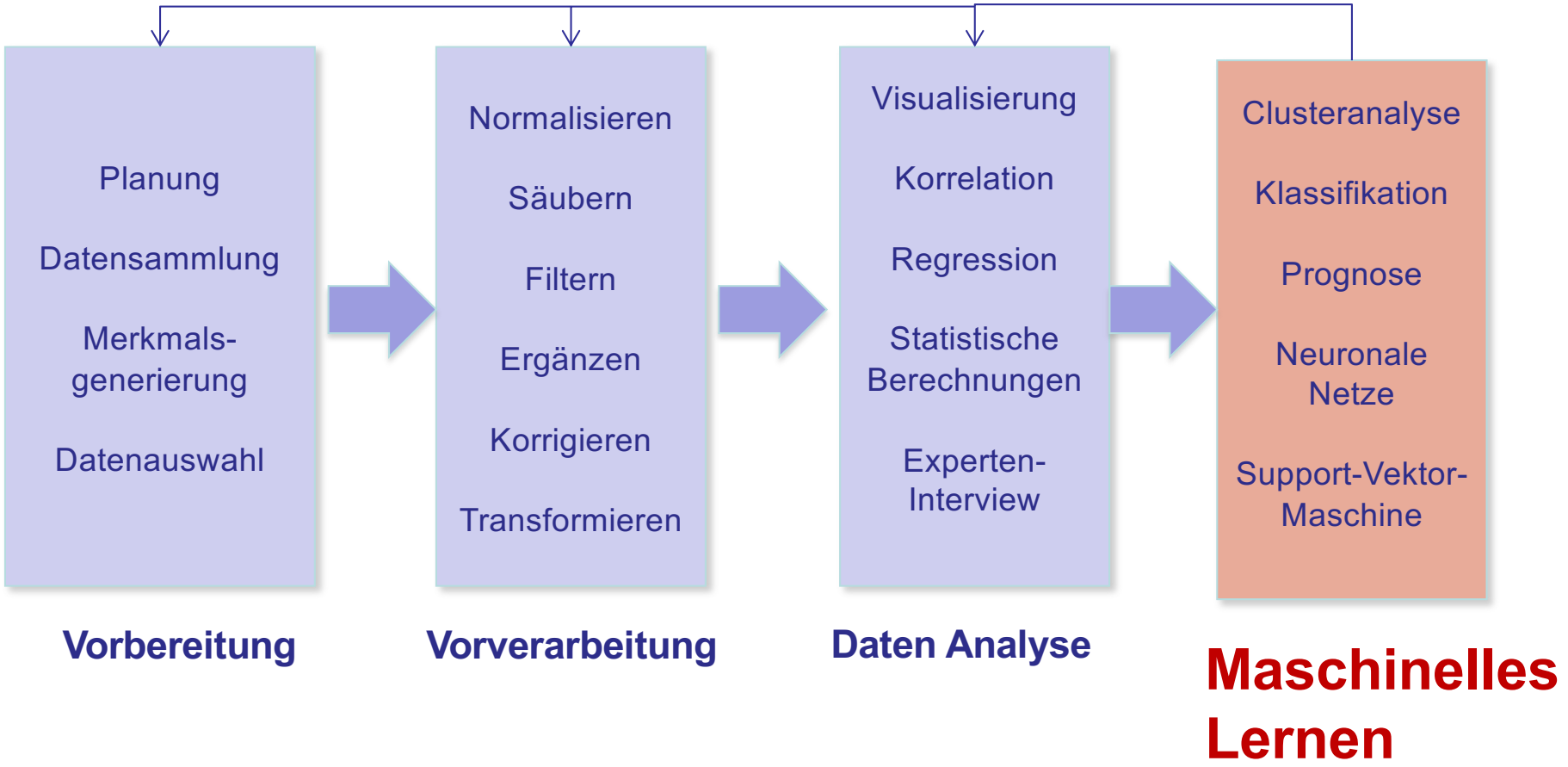
Dozentin: Grit Behrens
mailto: grit.behrens@hsbi.de

Studiengang Informatik Fachbereich Campus Minden

Lehrinhalte heute

1. Einstieg zu Datenanalyse und Einführung in Maschinelles Lernen
2. Datenanalyse mit Python
3. **Maschinelles Lernen und Neuronale Netze**

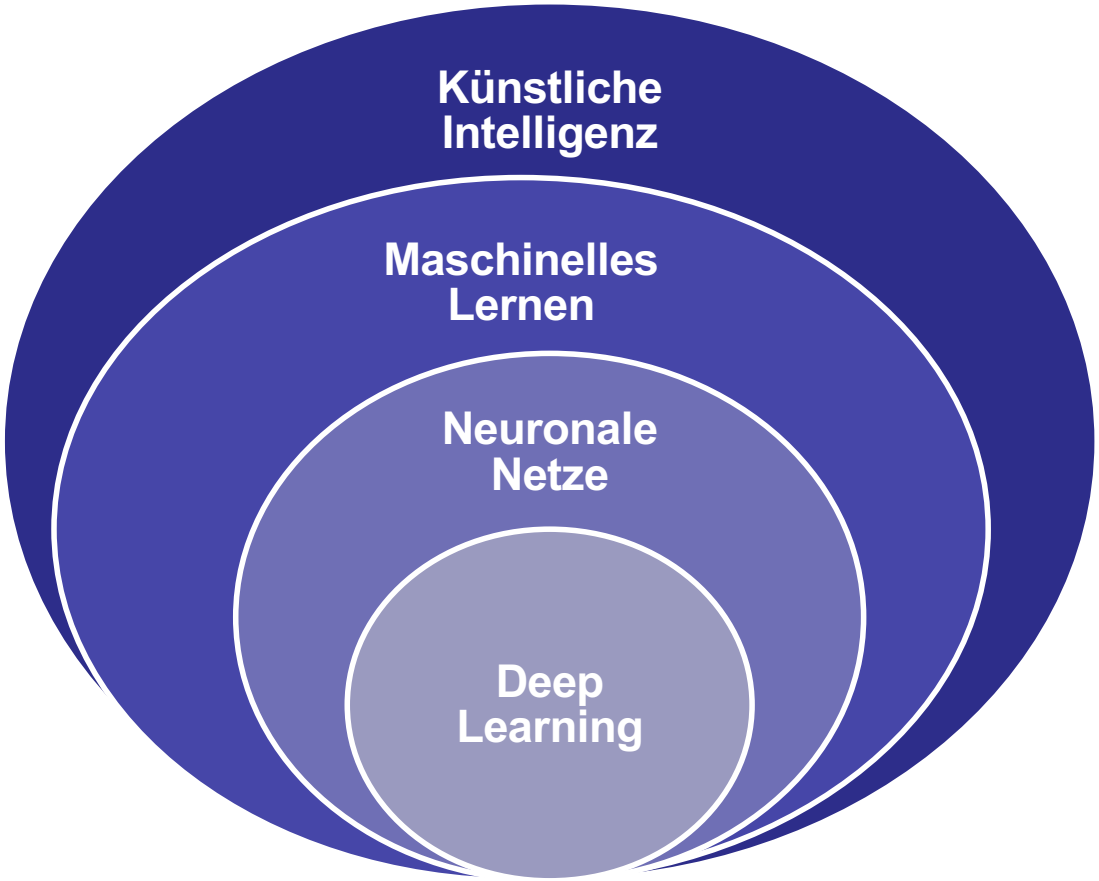
Das Data-Mining ist ein Prozess



Maschinelles Lernen

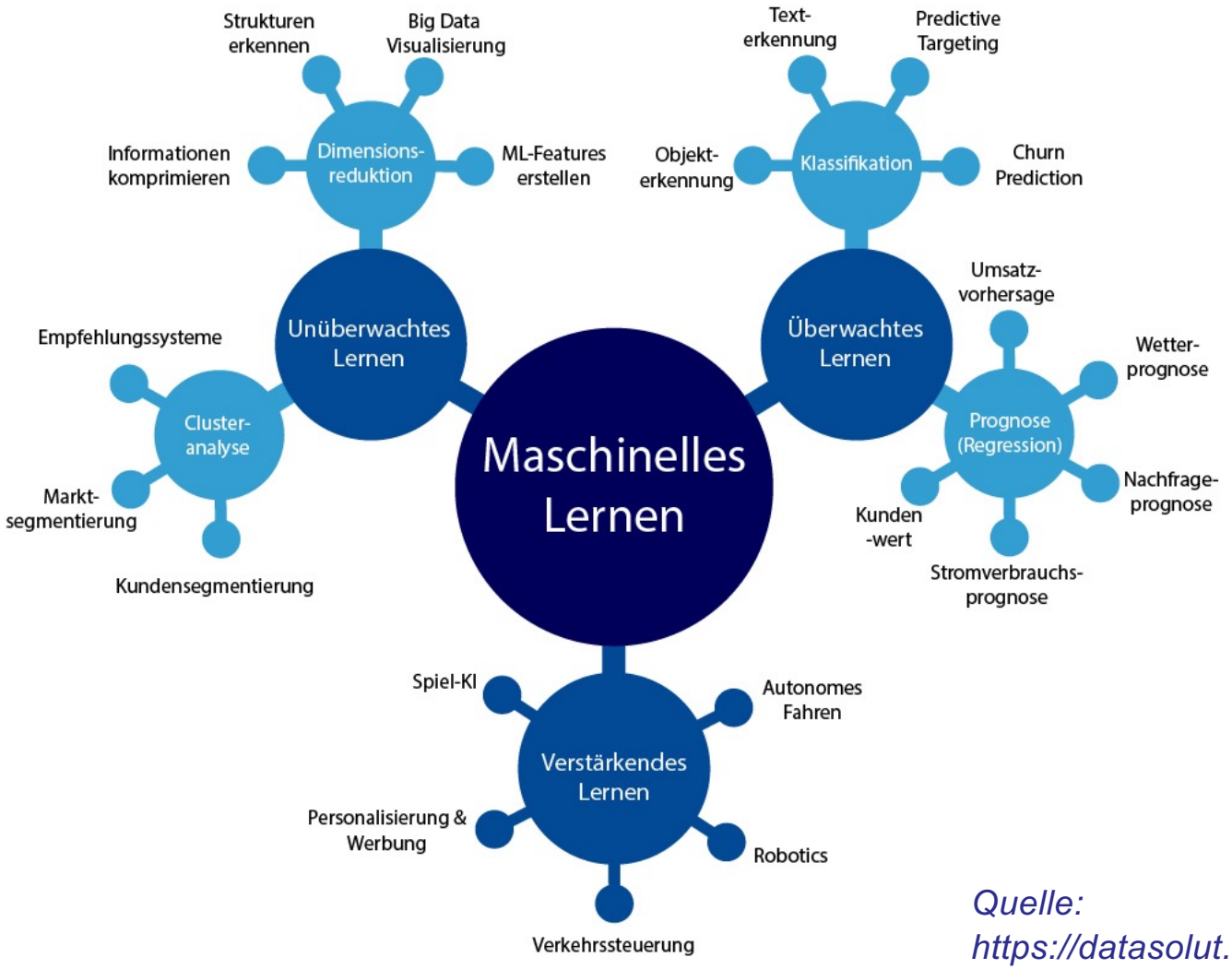
- **Einordnen der Begriffe**
- Clusteranalyse mit K-means
- K-means Clustering mit `Scikit-Learn`
- Klassifikation mit Neuronalen Netzen
- Datensätze für Training und Validierung
- Korrektheitsmaße
- NN-Klassifikation mit `Scikit-Learn`

Maschinelles Lernen : Einordnen der Begriffe KI, ML, DL, NN



Quelle *https://kirevolution.com/machine-learning-was-ist-und-wie-funktioniert-das-maschinelle-lernen*

Arten von Maschinellem Lernen



Quelle: https://datasolut.com/machine-learning-vs-deep-learning/

Maschinelles Lernen

- Einordnen der Begriffe
- **Clusteranalyse mit K-means**
- K-means Clustering mit `Scikit-Learn`
- Klassifikation mit Neuronalen Netzen
- Datensätze für Training und Validierung
- Korrektheitsmaße
- NN-Klassifikation mit `Scikit-Learn`

K-means Clustering - Theorie

Harter Partitionierungsalgorithmus: ***jedes Element*** des Datensatzes soll ***genau einem Cluster*** zugeordnet werden.

- ***n Elemente X*** sollen in k disjunkte Cluster c_i, ***i=1,...,k*** aufgeteilt werden
- Die Cluster werden durch den Mittelwert ***mean*** μ_i der Elemente im Cluster ***i*** charakterisiert -> Clusterschwerpunkt oder ***„Clustercentroid“***
- Minimierungsproblem des Algorithmus:

$$\arg min_c \sum_{j=1}^{k} \sum_{x \in c_j} d(x, \mu_j) = \arg min_c \sum_{j=1}^{k} \underbrace{\sum_{x \in c_j} \|x - \mu_j\|_2^2}$$

Summe der Abstandsquadrate (Euklidischer Anstand)

-> Die Summe der Abstandsquadrate oder der Euklidischen Abstände aller Elemente zum jeweiligen Centroidenmittelpunkt gerechnet über alle Cluster

- Unüberwachtes oder überwachtes Lernen?

K-means Clustering - Algorithmus

Iterativer Algorithmus mit 4 Schritten: sucht nach einer Lösung für das Minimierungsproblem.

1. ***Anzahl der k Cluster*** wird festgelegt
2. ***n Elemente*** werden den jeweils nächsten Cluster-Centroiden μ_i zugeordnet
3. Berechnen der ***k*** neuen Clustercentroiden c_j
4. ***Falls*** keins der ***n*** Elemente seine Zugehörigkeit gewechselt hat, dann **EXIT**
   Andernfalls : gehe zu Schritt 2

Es werden verschiedene Verfahren zur Wahl der initialen Centroiden angewendet.

Was würde Ihnen einfallen?

- ...
- ...
- ....

K-means++ Algorithmus für initiale Clusterzentroiden

Der k-Means++-Algorithmus wählt die initialen Cluster-Schwerpunkte nach folgender Vorschrift:

1. Wähle als ersten Cluster-Schwerpunkt zufällig ein Objekt aus.

2. Für jedes Objekt berechne den Abstand ***D(x)*** zum nächstgelegenen Cluster-Schwerpunkt

3. Wähle ***zufällig*** als nächsten Cluster-Schwerpunkt ein Objekt aus. Die ***Wahrscheinlichkeit, mit der ein Objekt ausgewählt wird, ist proportional zu*** $D^2(x)$,d. h. je weiter das Objekt von den bereits gewählten Cluster-Schwerpunkten entfernt ist, desto wahrscheinlicher ist es, dass es ausgewählt wird.

4. Wiederhole Schritt 2 und 3 bis ***k*** Cluster-Schwerpunkte bestimmt sind

5. Führe nun den üblichen k-Means Algorithmus aus

https://de.wikipedia.org/wiki/K-Means-Algorithmus

Maschinelles Lernen

- Einordnen der Begriffe
- Clusteranalyse mit K-means
- **K-means Clustering mit `Scikit-Learn`**
- Klassifikation mit Neuronalen Netzen
- Datensätze für Training und Validierung
- Korrektheitsmaße
- NN-Klassifikation mit `Scikit-Learn`

K-means Clustering – mit Scikit-Learn

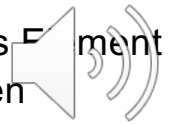
```
from sklearn.datasets import load_iris          #Einbinden der Bibliotheken
from sklearn.cluster import Kmeans              # Bibliothek für Kmeans-Algorithmus aus sklearn
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

iris = load_iris()                              # Iris-Datensatz laden
df = pd.DataFrame(iris.data, columns=iris.feature_names)    # Dataframe erstellen
x= iris.data                                    # Extrahieren der numerischen Werte

kmeans = KMeans(n_clusters=3,init = 'k-means++',   max_iter = 100, n_init = 10,
random_state = 0)                               #K-Means-Klassifikation initialisieren
y_kmeans = kmeans.fit_predict(x)                #K-means durchführen auf IRIS-Daten
```

`n_clusters=3`	Anzahl der Cluster ist 3
‘`k-means++’`	intelligente Auswahl der initialen Cluster Centroiden – z.B. auch möglich ‚random‘
`max_iter = 100,`	Maximale Anzahl der Interationen , default=300
`random_state = 0,`	legt die Anzahl Zufallszahlgenerierung für die initialen Clustercentroiden fest. (0- nicht gewählt)
`n_init = 10`	Anzahl der Iterationen für k-means++, falls gewählt, mit jeweils unterschiedlichen Clusterzentroiden

Methode `kmeans.fit_predict`

berechnet die Cluster Centroiden und bestimmt den Cluster Index für jedes Element
gibt die Indizees der Cluster zurück, zu denen die Elemente jeweils gehören

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

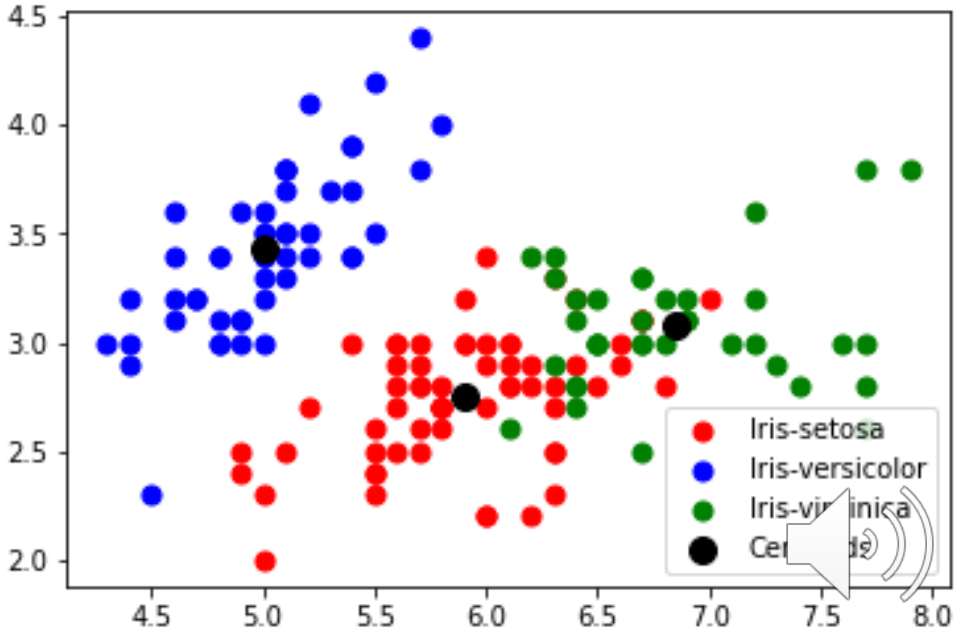


K-means Clustering – mit Scikit-Learn

```
...
print(kmeans.cluster_centers_)  #display cluster centers

plt.scatter(x[y_kmeans == 0,0], x[y_kmeans == 0,1],s = 100, c ='red',label =
'Iris-setosa')
plt.scatter(x[y_kmeans == 1,0], x[y_kmeans == 1,1],s = 100, c ='blue',label =
'Iris-versicolor')
plt.scatter(x[y_kmeans == 2,0], x[y_kmeans == 2,1],s = 100, c ='green',label =
'Iris-virginica')                    # Scatter-Plot für alle 150 Elemente , Achsen: 2 Features 0 und 1
                                     der drei Cluster eingefärbt mit IRIS-Art

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1],s = 100, c
= 'black', label = 'Centroids')    #plotting for the two first features for 3 centroids
plt.legend()
plt.show()
```

```
print(kmeans.cluster_centers_):
[[5.9016129 2.7483871 4.39354839
1.43387097] [5.006 3.428 1.462 0.246 ]
[6.85 3.07368421 5.74210526 2.07105263]]

print(y_kmeans):
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 2 0 2 2 2 2 0 2 2 2 2 2 2 0 0 2 2 2 2
0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0
2 2 2 0 2 2 2 0 2 2 0]
```

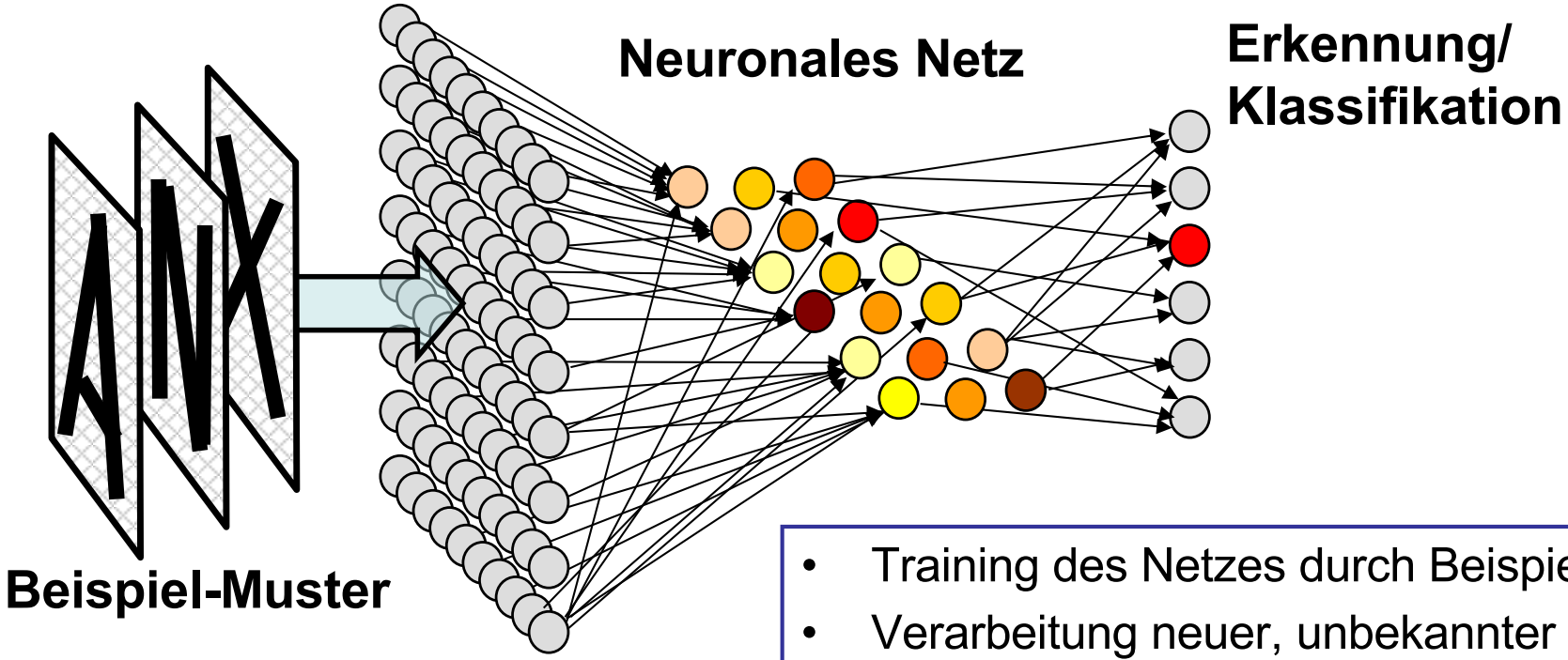
ML :



Maschinelles Lernen

- Einordnen der Begriffe
- Clusteranalyse mit K-means
- K-means Clustering mit `Scikit-Learn`
- **Klassifikation mit Neuronalen Netzen**
- Datensätze für Training und Validierung
- Korrektheitsmaße
- NN-Klassifikation mit `Scikit-Learn`

Einführung: Neuronale Netze lernen aus Beispielen



- Training des Netzes durch Beispiele
- Verarbeitung neuer, unbekannter Muster möglich
- Wissensrepräsentation

Einführung: Anwendungstechnologien

- ***Mustererkennung durch Klassifikation***
  - erst Lernverfahren (Training mit bekannten Beispielen)
  - danach Klassifikation mit eingestelltem Netzwerk

- ***Prognose***
  - Vorhersage von Zugehörigkeiten, Ereignissen und Prozessen aufgrund der Strukturierung als Klassifikationsproblem mit vorhergehendem Training

Künstliche Neuronale Netze

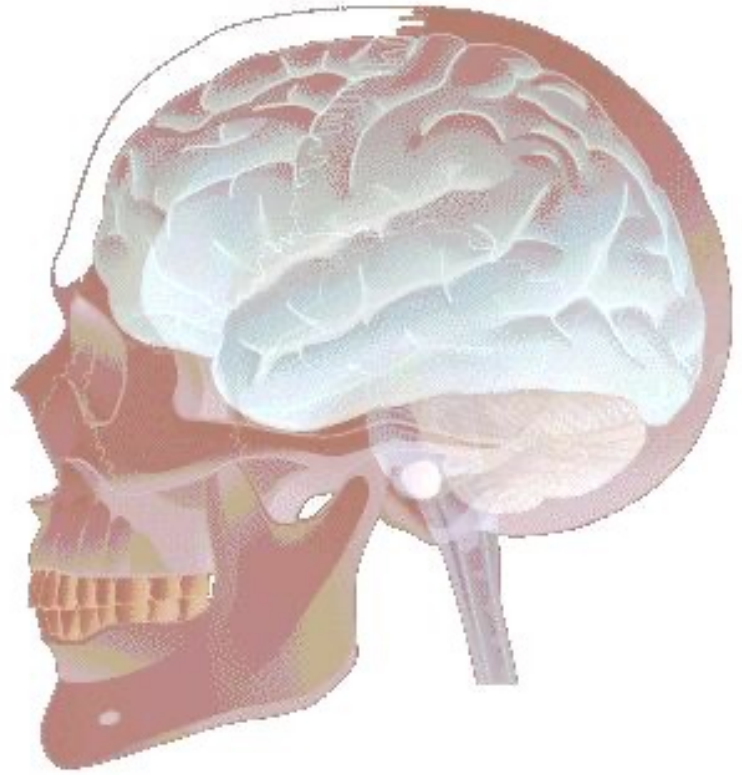
Der Mensch lernt durch Beispiele

oder auch “learning by doing”

- Nachahmen ist häufig durch Versuch und Irrtum geprägt.

Der Mensch nutzt sein Gehirn.

Das Gehirn besteht aus Millionen einzelner Zellen.
Eine Zelle ist mit Zehntausenden von anderen Zellen verbunden.



Künstliches Neuronales Netz:

Informationsverarbeitung in Analogie zur Arbeitsweise von Säugetiergehirnen
massiv parallele, lernfähige Systeme
große Anzahl einfach strukturierter Zellen

Künstliche Neuronale Netze

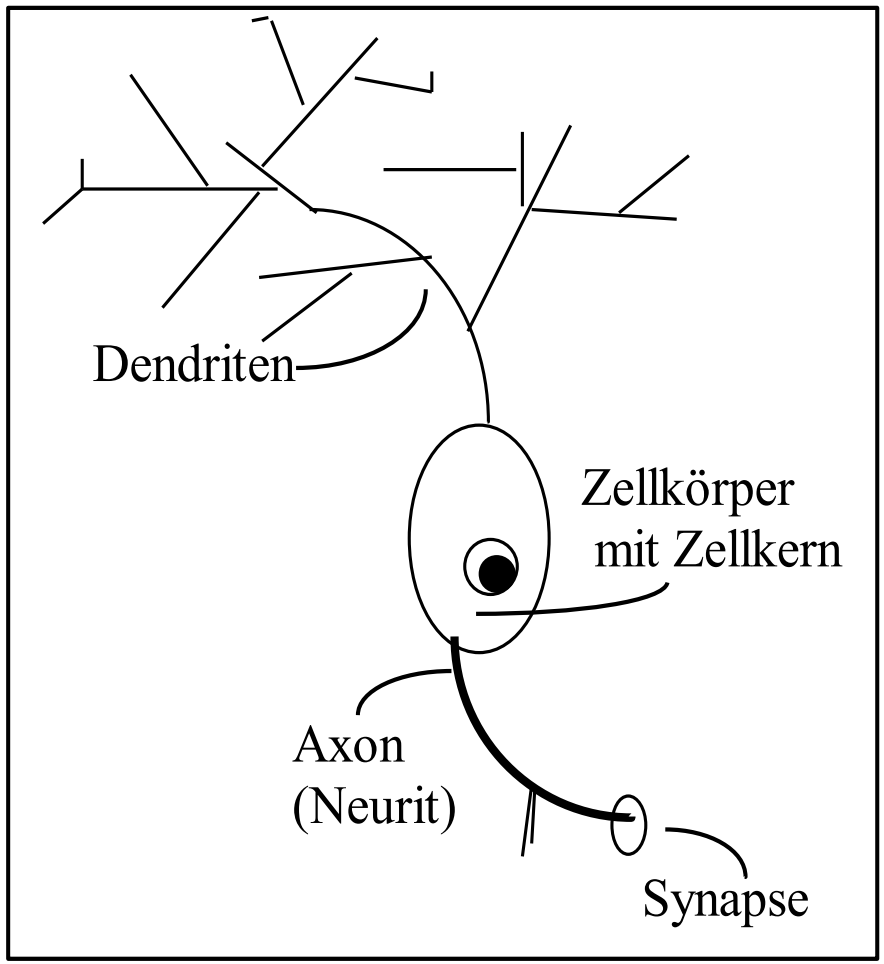
Neuronale Netze in der KI *versuchen die Arbeitsweise des menschlichen Gehirns nachzubilden.*

Millionen menschlicher Nervenzellen sind trainierbar und übernehmen Steuerungsaufgaben. Auch Computerprogramme sollen aus Beispielen Gelerntes auf neue Situationen anwenden können.

Zur Nachbildung im Computer benötigen wir:

- Software Neuronen
- Software Verbindungen zwischen Neuronen
- Software Lernalgorithmen

Ein biologisches Neuron



Dendriten:

- Eingang
- Aufnahme der Erregung

Axon:

- Ausgang,
- Weiterleitung der Erregung,
- 1mm-1m lang

Synapse:

- Übergabe der Erregung: an andere Zellen über Dendriten anderer Neuronen
- eine Zelle besitzt ca. 1.000 bis 10.000 Verbindungen mit anderen Zellen

Zellkern:

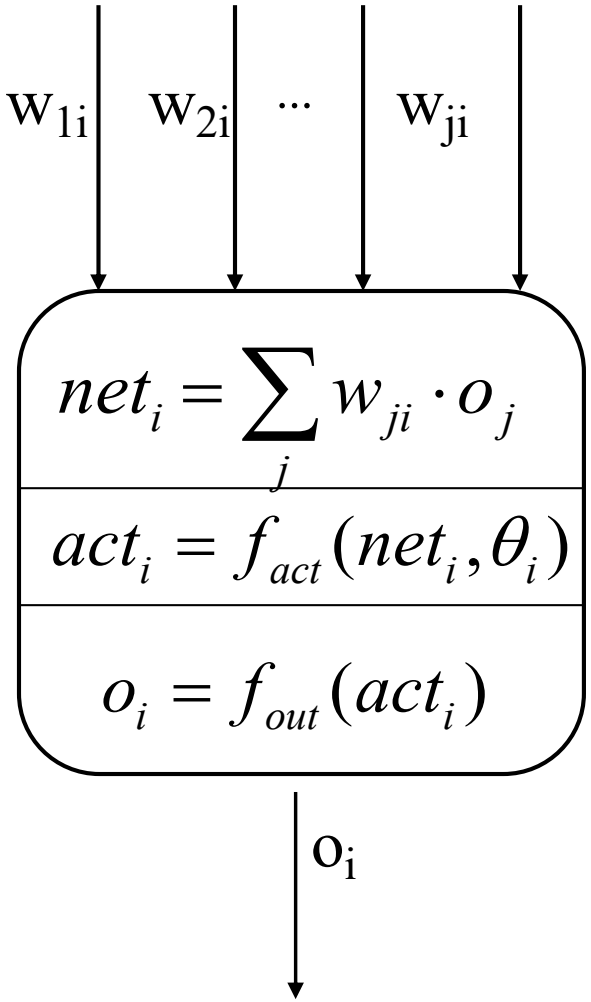
- Verarbeitung
- Bestimmen der Erregung

Das künstliche Neuron

Intuitive Beschreibung:
*Ein **künstliches Neuron** ist eine am biologischen Vorbild orientierte einfache Verarbeitungseinheit mit einem kleinen lokalen Speicher.*

Dendriten:	gewichtete Verbindungen Gewicht: reelle Zahl
Axon:	Ausgabe: reelle Zahl
Synapsen:	Identität: Ausgabe wird direkt weiter geleitet (Ausgabefkt.)
Zellkern:	Einheit mit einfachen Verarbeitungsfunktionen **Eingabe** = (meist viele) reelle Zahlen **Verarbeitung** = Bestimmen der Erregung **Ausgabe** = reelle Zahl, die prop. zur Erregung ist

Das künstliche Neuron



net : Netz-Eingabe, Propagierungsfkt.

w_{ij} : Gewicht einer Verbindung

o_j : Eingabe

act : Aktivierung

f_{act} : Aktivierungsfunktion

θ : Schwellwert

f_{out} : Ausgabe-Funktion (meist ID)

o_i : Ausgabe

Aktivierungsfunktionen

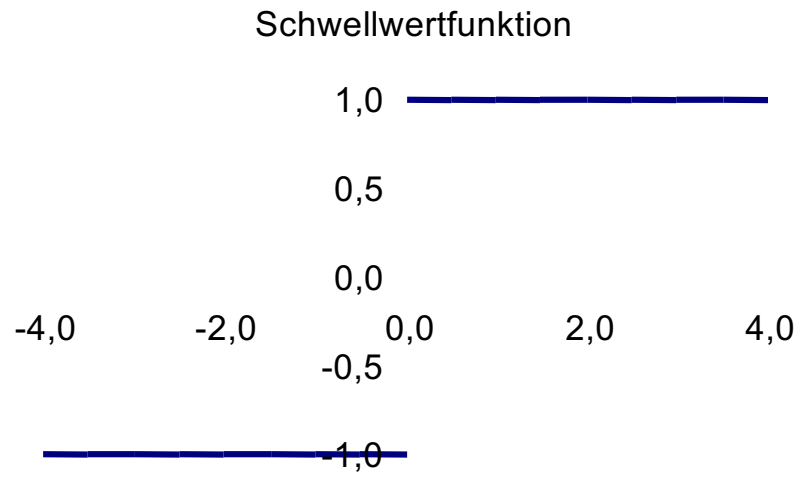
Die Aktivierungsfunktion *bestimmt aus der Netzeingabe unter Berücksichtigung eines Schwellwertes die neue Aktivität (den neuen Zustand) des Neurons. Sie weist meist* ***einen sigmoiden Charakter*** *auf.*

• *durch sigmoide Aktivierungsfunktionen können die eingehenden Werte in einen beschränkten Bereich der Aktivierungszustände abgebildet werden*

Konkrete Arten von Aktivierungsfunktionen:

- lineare Funktionen (z.B. Identität)
- Schwellwertfunktionen
- logistische Funktionen
- Tangens Hyperbolicus

Aktivierungsfunktionen

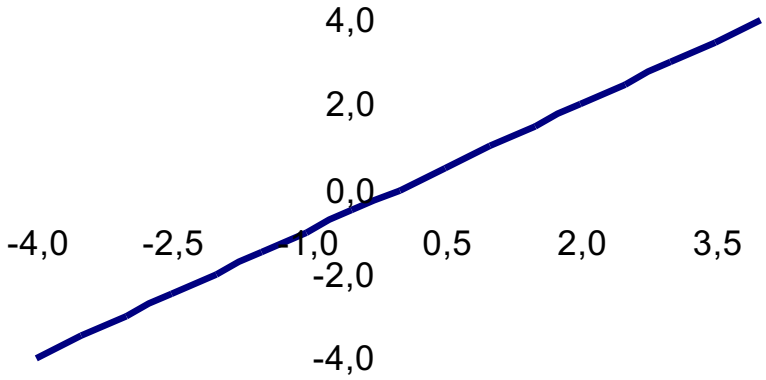


Schwellwertfunktion:

- **schärfste sigmoide Funktion**
- **übersteigt die Netzeingabe den Schwellwert, wird das Neuron aktiviert**

f(x) = $\begin{cases} \textbf{1, x} \geq \theta \\ \textbf{-1, sonst} \quad \textbf{/* oft auch 0*/} \end{cases}$

Lineare Funktion: Identität

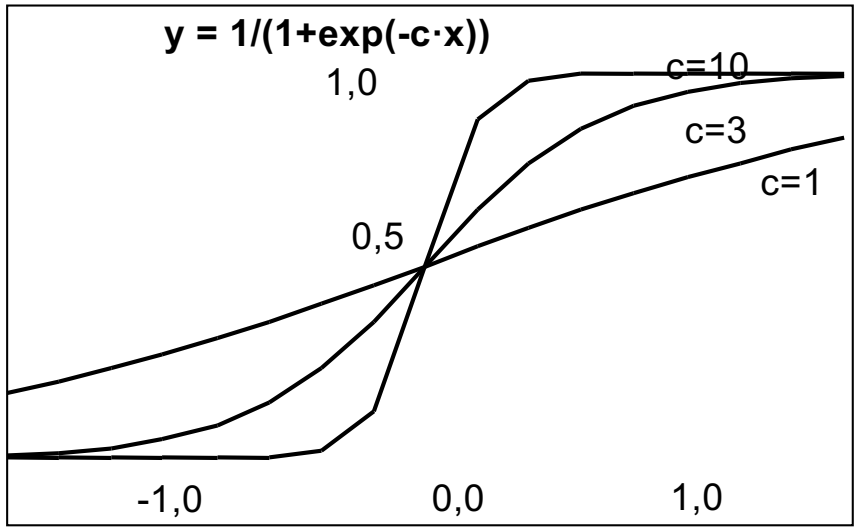


Lineare Funktion - Identität :

$$f(x) = x$$

- **reicht die Eingabewerte weiter**

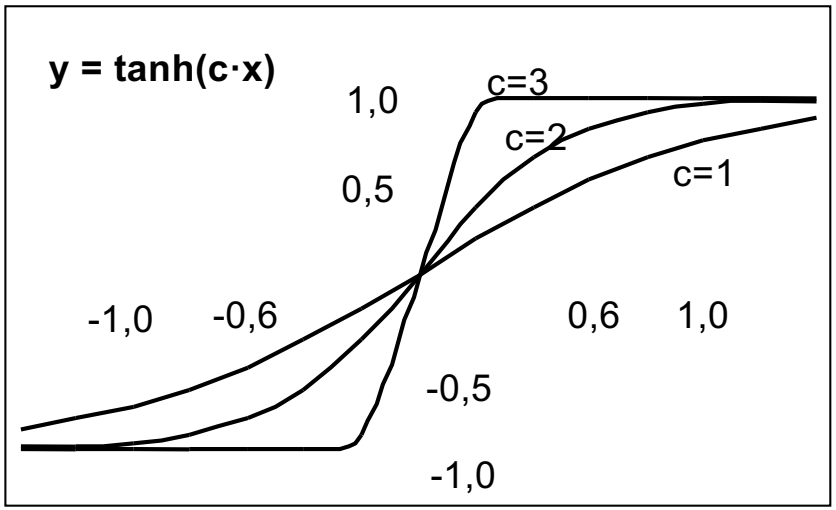
Aktivierungsfunktionen



Logistische Funktion:

$$f(x) = \frac{1}{1 + e^{-c \times x}}$$

- **Parameter *c* für Steilheit der Kurve beim Übergang von 0 auf 1**



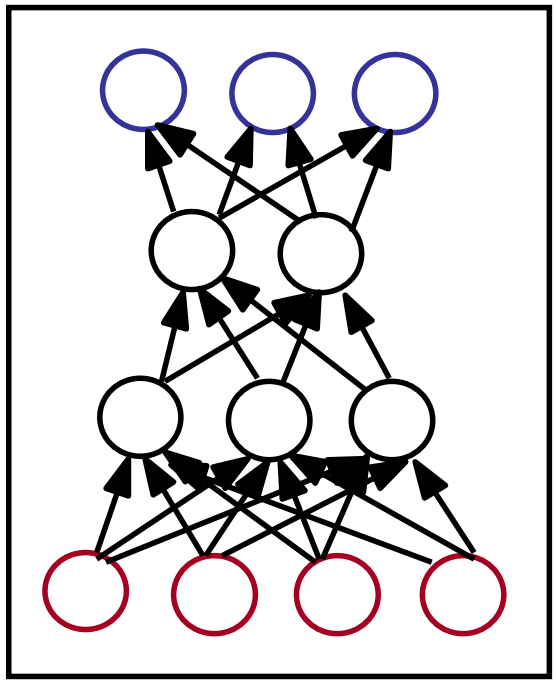
Tangens Hyperbolicus:

$$f(x) = tanh(c \times x)$$

- **ermöglicht Aktivierung des Neurons aus dem Bereich [-1, +1]**
- **Parameter *c* für Steilheit der Kurve beim Übergang von 0 auf 1**

Definition des Neuronalen Netzes

- im Netz unterscheidet man:
  - Eingabe-Schicht (input layer)
    Eingabe-Neuron **(rot)**
  - Ausgabe-Schicht (output layer)
    Ausgabe-Neuron **(blau)**
  - verdeckte Schicht (hidden layer)
    verdecktes Neuron **(schwarz)**
- n-stufiges Netz besitzt
  n trainierbare Verbindungsebenen
  - d.h. n+1 Schichten,
    davon sind n-1 verdeckt.



Frage: Wie groß ist n für das abgebildete Netz?

n = 3 ; 3-stufiges neuronales Netz mit 4 Schichten und davon 2 verdeckten Schichten

Definition des Neuronalen Netzes

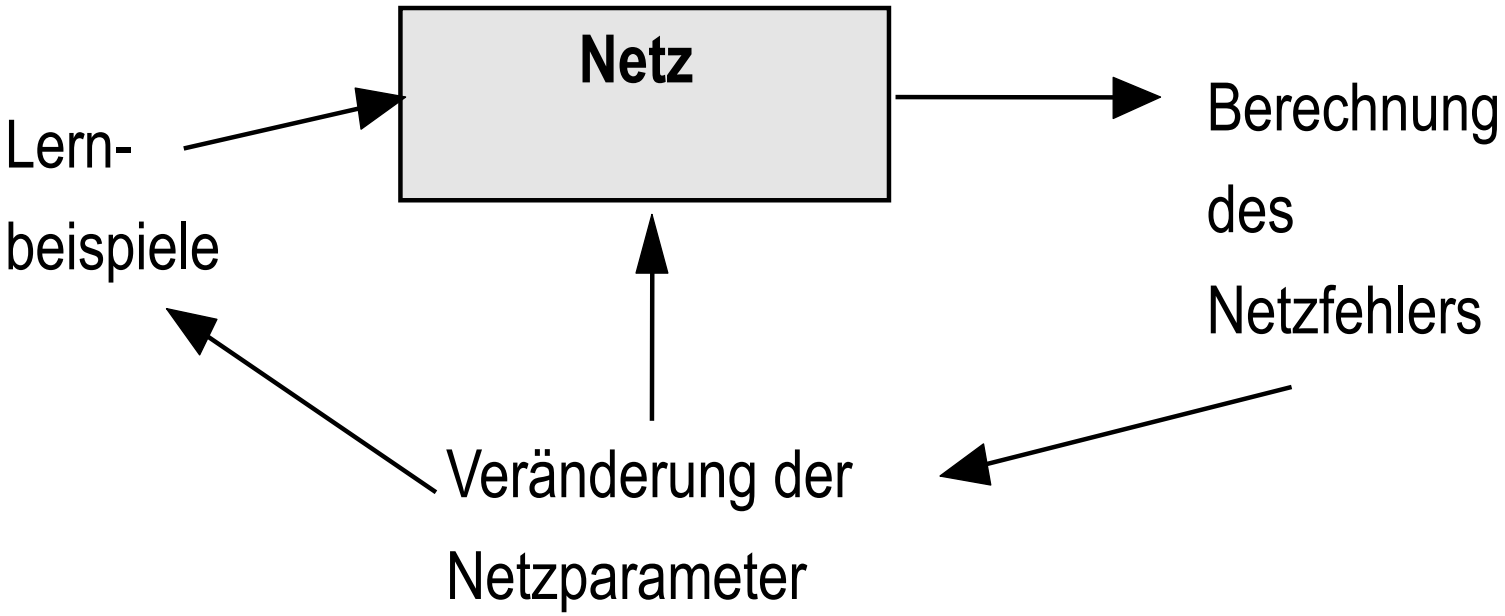
Ein Neuronales Netz *besteht aus einer Menge von Neuronen, die durch gerichtete und gewichtete Verbindungen miteinander verknüpft sind.*

- eine Verknüpfung zwischen zwei Neuronen **i und j** ist durch das Verbindungsgewicht $\mathbf{w_{ij}} \neq \mathbf{0}$ definiert.
- Verbindungsgewichte lassen sich als ***Matrix*** darstellen

- Netze können auch als *gerichteter Graph* dargestellt werden

- Die Menge und Anordnung der Verbindungen und Schichten bestimmt die *unterschiedlichen Architekturen*

Lernen in NN



- ***Lernen erfolgt aus Beispielen***

Lernen - Mögliche Anpassungen der Netzparameter:

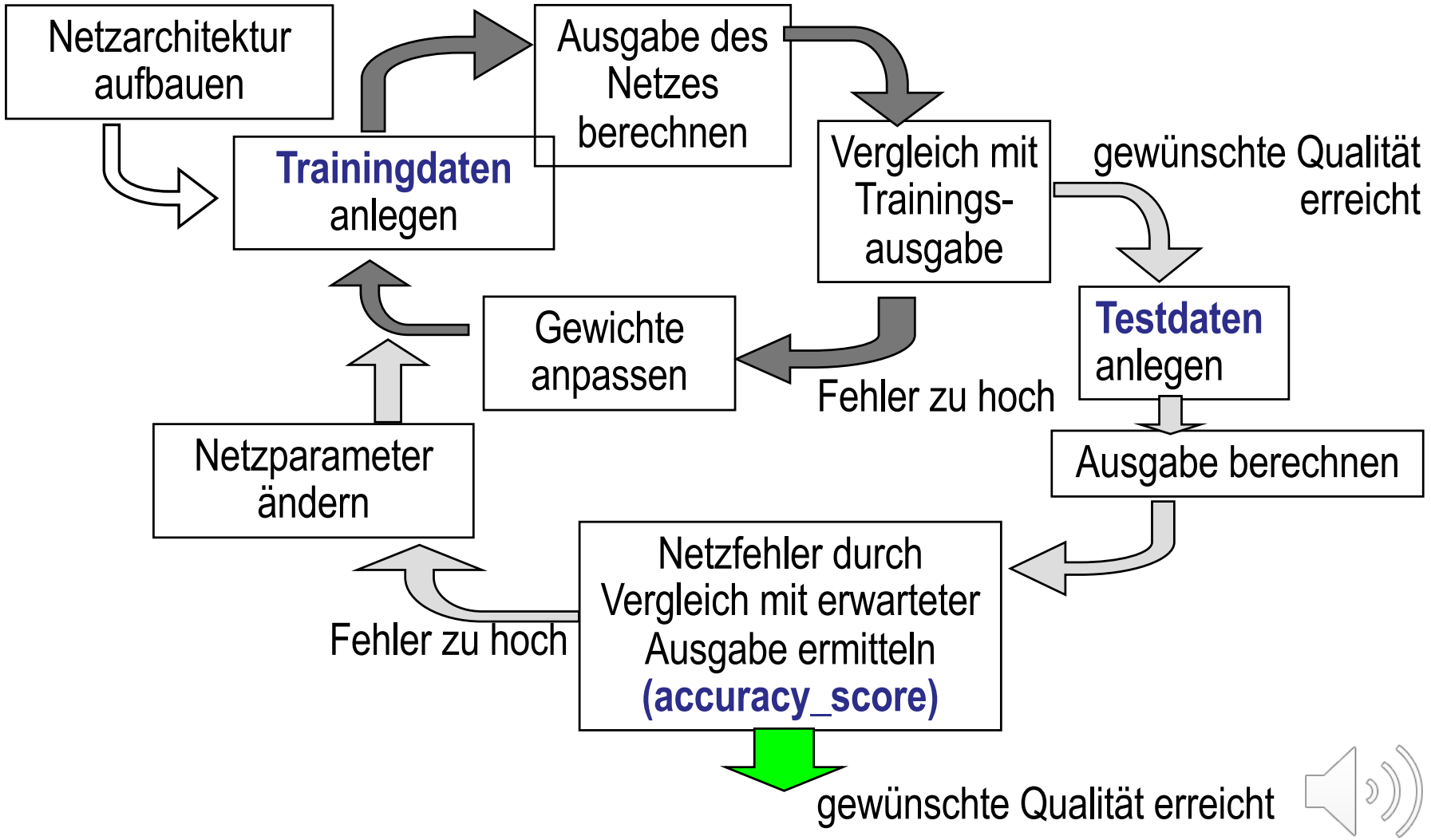
1. **Modifikation der Stärke von Verbindungen**
   – wird am häufigsten verwendet
2. **Entwicklung neuer Verbindungen**
3. **Löschen existierender Verbindungen**
4. **Modifikation der Schwellwerte von Neuronen**
5. **Modifikation der Funktionen**
   (Aktivierungs-, Propagierungs-, Ausgabe-)
6. **Entwicklung neuer Neuronen** (z.B. *Genetic Neural Network)*
7. **Löschen von Neuronen**
8. **Nutzen von vortrainierten Netzen**

   ...

Einige Lernregeln

- ***Hebbsche Lernregel***
  - “Wenn ***Zelle j*** eine Eingabe von ***Zelle i*** erhält und beide gleichzeitig stark aktiviert sind, dann erhöhe das Verbindungsgewicht w_{ij}.”
- ***Delta-Regel***
  - Gewichtsänderung proportional zur *Differenz* aus *erwarteter und tatsächlicher* Aktivierung
- ***Backpropagation-Regel***
  - Anwendung der Delta-Regel für Netze mit *mehr als einer* trainierbaren *Schicht*

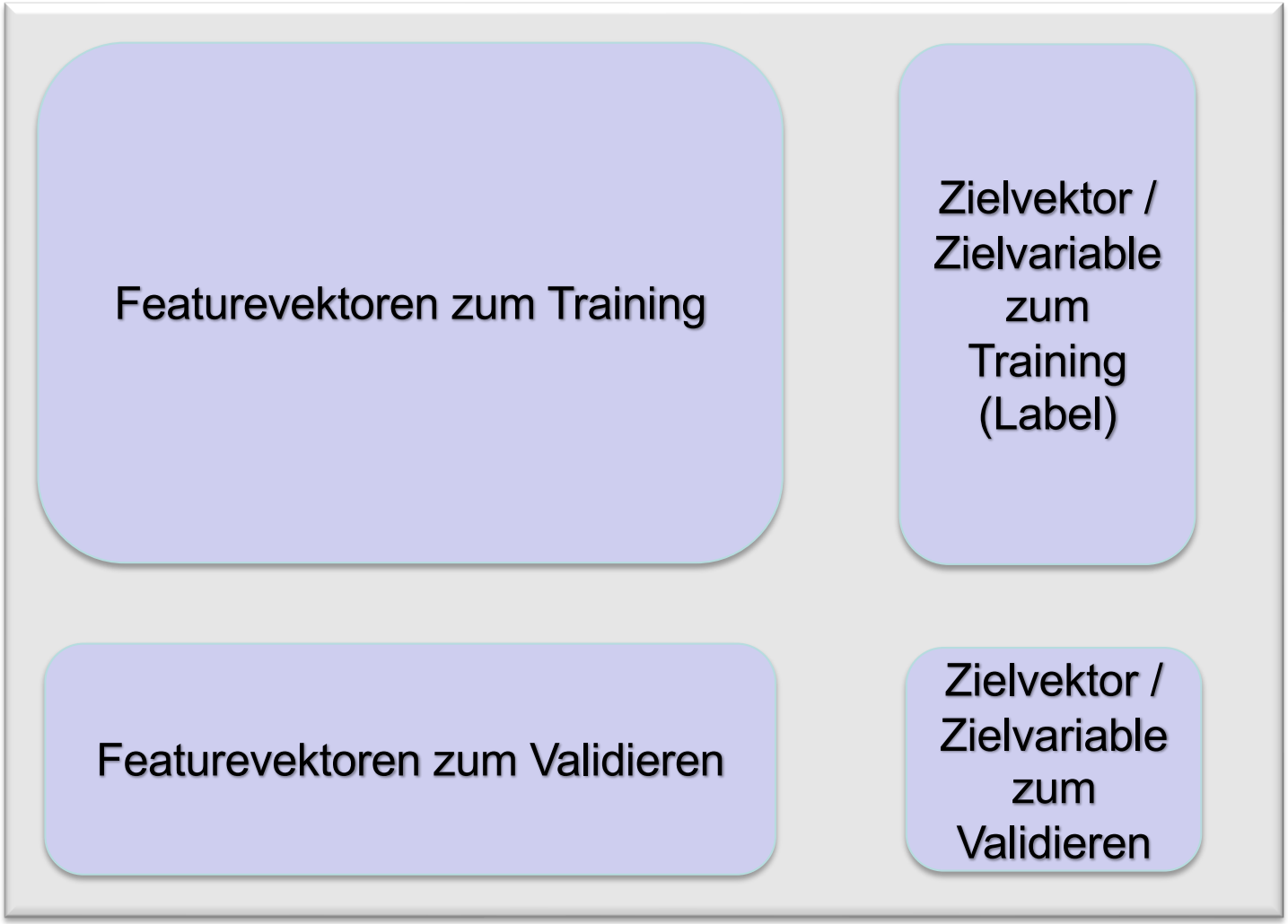
Entwicklung neuronaler Netze



Maschinelles Lernen

- Einordnen der Begriffe
- Clusteranalyse mit K-means
- K-means Clustering mit `Scikit-Learn`
- Klassifikation mit Neuronalen Netzen
- **Datensätze für Training und Validierung**
- Korrektheitsmaße
- NN-Klassifikation mit `Scikit-Learn`

Aufteilung der Daten in gelabelte Lern- und Testdatensätze



Aufteilung der Daten in Lern- und Testdatensatz für IRIS-Daten

Aufteilung der Daten in Lern- und Testdatensatz für IRIS-Daten

	sepal.length	sepal.width	petal.length	petal.width
22	4.6	3.6	1.0	0.2
15	5.7	4.4	1.5	0.4
65	6.7	3.1	4.4	1.4
11	4.8	3.4	1.6	0.2
42	4.4	3.2	1.3	0.2
...	...	...	...	...
71	6.1	2.8	4.0	1.3
106	4.9	2.5	4.5	1.7
14	5.8	4.0	1.2	0.2
92	5.8	2.6	4.0	1.2
102	7.1	3.0	5.9	2.1

```
0
0
1
0
0
..
1
2
0
1
2
```

	sepal.length	sepal.width	petal.length	petal.width
73	6.1	2.8	4.7	1.2
18	5.7	3.8	1.7	0.3
118	7.7	2.6	6.9	2.3
78	6.0	2.9	4.5	1.5
76	6.8	2.8	4.8	1.4

```
1
0
2
1
1
```

Aufteilung der Daten in gelabelte Lern- und Testdatensätze

- Daten werden in ***Trainings- und Testdatensatz aufgeteilt***
  - Häufig 30% zu 70%
  - oder 20% zu 80%

- Trainings- und Testdaten sollen beide ***repräsentativ*** sein

- Testdaten sollen ***unabhängig*** von den Trainingsdaten sein (disjunkte Mengen)
  - Die Testdaten dürfen bei dem Training nicht genutzt werden!

- Testdaten sollten die ***gleiche Wahrscheinlichkeitsverteilung*** wie der Trainingsdatensatz aufweisen.

-> das trainierte Modell kann erfolgreich auf unbekannte Daten zur Prognose angewandt werden.

Maschinelles Lernen

- Einordnen der Begriffe
- Clusteranalyse mit K-means
- K-means Clustering mit `Scikit-Learn`
- Klassifikation mit Neuronalen Netzen
- Datensätze für Training und Validierung
- **Korrektheitsmaße**
- NN-Klassifikation mit `Scikit-Learn`

Richtigkeit eines Klassifikators (accuracy)

Def. Richtigkeit:
Anzahl der korrekt prognostizierten Merkmalsvektoren geteilt durch die Anzahl aller Merkmalsvektoren im ***Validierungsdatensatz.***

$$acc = \frac{Anzahl\ der\ korrekt\ klassifizierten\ Merkmalsvektoren}{n}$$

n – Anzahl der Merkmalsvektoren

Bei binärer Betrachtungsweise: Korrekt klassifizierte Merkmalsvektoren können die richtigen positiven und die richtigen negativen Zuordnungen sein.

Richtig Positive (TP): Der Klassifikator prognostiziert einen Merkmalsvektor als positiv, was er auch in Wirklichkeit ist.
Falsch Positive (FP): Der Klassifikator prognostiziert einen Merkmalsvektor als positiv, er ist aber in Wirklichkeit negativ.
Richtig Negative (TN): Der Klassifikator prognostiziert einen negativen Merkmalsvektor richtig als negativ, denn im Wirklichkeit ist er auch negativ.
Falsch Negative (FN): Der Klassifikator prognistiziert einen negativen Merkmalsvektor fälschlicherweise, in Wirklichkeit ist er positive.

Wahrheitsmatrix (confusion matrix)

TP	**FP**
FN	**TN**

***Prognostizierte Klasse (P/N)**

***Tatsächliche Klasse (T/F)**

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

*Accuracy - Korrektheit

$$Genauigkeit = \frac{TP}{TP + FP}$$

$$negative\ Erkennungsleistung = \frac{TN}{TN + FN}$$

Maschinelles Lernen

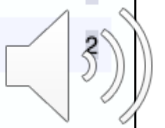
- Einordnen der Begriffe
- Clusteranalyse mit K-means
- K-means Clustering mit `Scikit-Learn`
- Klassifikation mit Neuronalen Netzen
- Datensätze für Training und Validierung
- Korrektheitsmaße
- **NN-Klassifikation mit `Scikit-Learn`**

NN-Klassifikation mit scikit-Learn

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split #libraries einbinden
from sklearn.neural_network import MLPClassifier

df = pd.read_csv('iris.csv')                    # Iris-Datensatz als DataFrame einlesen

variety_to_int = {'Setosa': 0, 'Versicolor': 1, 'Virginica': 2}
df['variety'] = df['variety'].map(variety_to_int) # Umwandeln der species-namen in int
                                                  # Schreiben der int in die letzte Spalte
df   #Ausgabe des DataFrame
```

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns



NN-Klassifikation mit scikit-Learn

```
....

X = df.iloc[:,:-1]       # Auswahl alle Zeilen, alle Spalten ohne die letzte (4 Merkmale als float)
y = df.iloc[:,-1]        #  Auswahl alle Zeilen, die letzte Spalte (spezies als int , label)
print('X:', X.shape,' y:', y.shape)       #Ausgabe X: (150,4) y: (150,)
...
```

X - Featurevektor

	sepal.length	sepal.width	petal.length	petal.width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

y- Targetvektor

```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: variety, Length: 150, dtype: int64
```

NN-Klassifikation mit scikit-Learn

```
...
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
                    #Aufteilen der 150 Vektoren in 120 Vektoren Lern- und 30 Vektoren  Testdatensatz

print('X_train:', X_train.shape, ' y_train:', y_train.shape)
print('X_test:', X_test.shape, ' y_test:', y_test.shape)
                                  # output: X_train: (120, 4) y_train: (120,) X_test: (30, 4) y_test: (30,)
```

`Sklearn.modelselection.train_test_split`

Teilt Elemente des Datensatzes mit zufälliger Auswahl in Trainings- und Testdatensatz.

Eingabeparameter :

- zwei gleich lange indexierbare Sequenzen (DataFrames, Listen, numpy arrays....) , hier: X,y
- test_size= 0.2; prozentualer Anteil des Testdatensatzes zw. 0.0 und 1.0 hier: 0.2 – 20 % der Merkmalsvektoren sollen im Testdatensatz sein
- random_state = 42 , steuert die Randomisierung integer= 42 ermöglicht Reproduktion

NN-Klassifikation mit scikit-Learn

```
...

clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(3, 3),
random_state=1)                          # MultiLayer - Perceptron  Klassifikator (NN) instanziieren

clf.fit(X_train, y_train)     # Model trainieren auf den Trainingsdaten mit
                              # Eingabe featurevektor X-train und
                              # Targetvektor (label) y-train
...
```

class **`sklearn.neural_network.MLPClassifier()`**

Multi-Layer-Perceptron Klassifikator

mit Eingabeparametern - **Hyperparameter**:

- Solver **`solver='lbfgs'`** – Optimierungsalgorithmus für die Verbindungsgewichte zwischen den Neuronen, findet lokales Minimum ; **Limited-memory Broyden–Fletcher–Goldfarb–Shanno**.
- **`alpha= 1e-5`** – Strafterm , defaultWert 0.0001
- **`hidden_layer_sizes=(3, 3)`** – **#** zwei hidden layer Größe 3
- **`random_state=1`** – determiniert die Randomisierung der initialen Verbindungsgewichte

NN-Klassifikation mit scikit-Learn

```
...
prediction = clf.predict(X_test) # Anwenden des  Modells und Generieren von Vorhersagen
                                 # auf dem Testdatensatz, Ergebnisse in prediction
print(prediction)
print(y_test.values)

acc_score = accuracy_score(y_test, prediction) #vergleicht Label der Testdaten (30
                                 # Elemente) mit vorhergesagten Werten und berechnet accuracy
print('Accuracy:', acc_score)            # alle Prognosen sind richtig! ->100% accuracy=1
```

```
Output:
[1 0 2 1 1 0 1 2 2 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
[1 0 2 1 1 0 1 2 2 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
 Accuracy: 1.0
```

Lehrinhalte nächstes Mal

1. Einstieg zu Datenanalyse und Einführung in Maschinelles Lernen
2. Datenanalyse mit Python
3. Maschinelles Lernen und Neuronale Netze
4. **Zeitreihen**

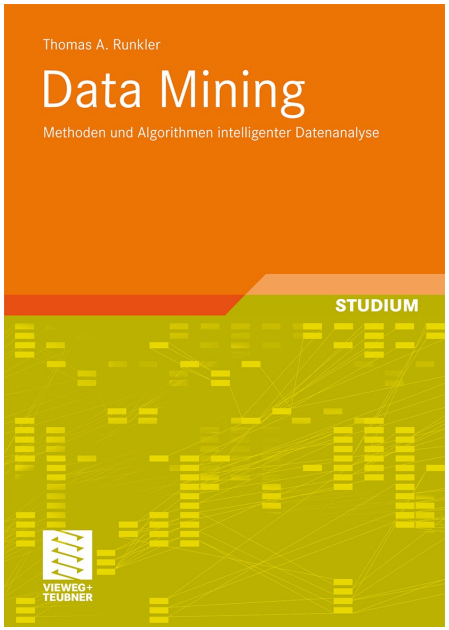
Literaturangaben und Quellen

Jörg Fochte „Maschinelles Lernen – Grundlagen und Algorithmen in Python

Carl Hanser Verlag (11/2020); ISBN-13 : 978-3446461444
3. Auflage 2020

Carl Hanser Verlag (01/2019); ISBN-13: 978-3-446-45996-0
2. Auflage 2019
Carl Hanser Verlag (08/2018); ISBN-13: 978-3-446-45291-6
1. Auflage 2018

Literaturangaben und Quellen



Thomas A. Runkler
„Data Mining – Methoden und Algorithmen intelligenter Datenanalyse“

2. Auflage:
Verlag: Springer-Verlag GmbH
Seitenzahl: 145
Erscheinungstermin: 5. Oktober 2015 Deutsch
ISBN-13: 9783834821713

https://www.python-kurs.eu/maschinelles_lernen.php
https://www.grund-wissen.de/informatik/python/scipy/pandas.html