

# DATA SCIENCE 1

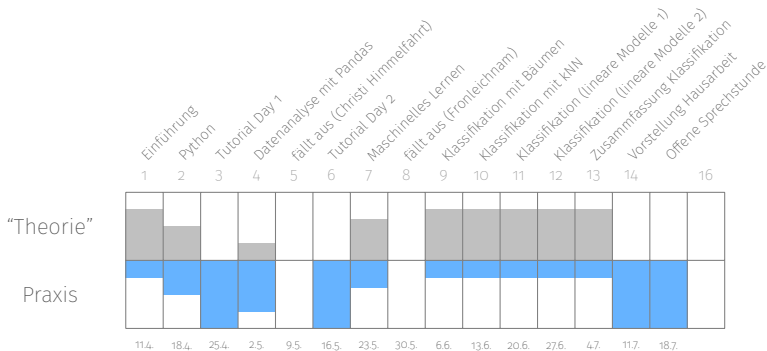
VORLESUNG 9

PROF. DR. CHRISTIAN BOCKERMANN

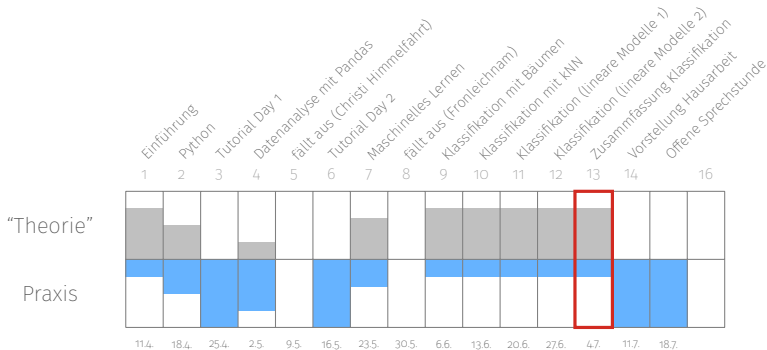
HOCHSCHULE BOCHUM

SOMMERSEMESTER 2024

## Wo sind wir heute?



## Wo sind wir heute?



- 1 Zusammenfassung Klassifikation
- 2 Maschinelles Lernen mit Python
- 3 Klassifikationsverfahren
- 4 Neuronale Netze (und ChatGPT)
- 5 Model Selection
- 6 Organisatorisches / Wie geht's weiter?
- 7 Was erwartet Sie in Data Science 2?

# Zusammenfassung Klassifikation

Lern-Algorithmen erwarten Daten häufig in Form einer Tabelle:

$d$ Merkmale					
ID	$a_1$	$a_2$	$\dots$	$a_d$	$y$
1	0	0	$\dots$	1	-1
2	0	1	$\dots$	1	+1
3	1	0	$\dots$	1	-1

$$\begin{aligned}\text{Beispiel } \mathbf{x}_2 &= (x_{a_1}, x_{a_2}, \dots, x_{a_d}, y) \\ &= (0, 1, \dots, 1, +1)\end{aligned}$$

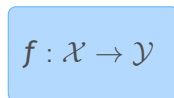
- Beispiele werden auch *examples* oder *instances* genannt
- Merkmale (engl. *features*) werden auch *attributes* oder *Variablen* (Statistik) bezeichnet

$a_1$	$a_2$	...	$a_d$	$y$
0	0	...	1	-1
0	1	...	1	+1
1	0	...	1	-1

Trainingsdaten  $\mathbf{X}, \mathbf{y}$ Algorithmus/  
Optimierung $f: \mathcal{X} \rightarrow \mathcal{Y}$ 

Modell

$a_1$	$a_2$	$\dots$	$a_d$	$y$
0	0	$\dots$	1	-1
0	1	$\dots$	1	+1
1	0	$\dots$	1	-1

Trainingsdaten  $\mathbf{X}, \mathbf{y}$ 

Modell

---

$a_1$	$a_2$	$\dots$	$a_d$	$y$
1	1	$\dots$	0	?

Neue Daten  $\mathbf{x}'$ ,  
 $\mathbf{y}$  unbekannt



$a_1$	$a_2$	...	$a_d$	$y$
0	0	...	1	-1
0	1	...	1	+1
1	0	...	1	-1

Trainingsdaten  $\mathbf{X}, \mathbf{y}$ Algorithmus/  
Optimierung

$$f: \mathcal{X} \rightarrow \mathcal{Y}$$

Modell

$a_1$	$a_2$	...	$a_d$	$y$
1	1	...	0	?

Neue Daten  $\mathbf{x}'$ ,  
 $\mathbf{y}$  unbekannt

Vorhersage

$$\hat{y} = f(\mathbf{x}')$$

## Klassifikation ordnet Beispielen diskreten Klassen zu

- Vorgegebene Klassen  $\mathcal{Y} = \{C_1, \dots, C_k\}$
- Gegeben Menge  $\mathbf{X} \times \mathbf{y} \subset \mathcal{X} \times \mathcal{Y}$  bei der jedem Beispiel  $\mathbf{x}_i$  die zugehörige Klasse zugeordnet ist:  $(\mathbf{x}_i, \mathbf{y}_i)$
- Qualitätsfunktion  $q : (\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \rightarrow \mathcal{Y}) \rightarrow \mathbb{R}$

### Ziel:

- Finde Modell

$$f : \mathcal{X} \rightarrow \mathcal{Y},$$

das die Qualitätsfunktion optimiert.

## Klassifikation ordnet Beispielen diskreten Klassen zu

- Vorgegebene Klassen  $\mathcal{Y} = \{C_1, \dots, C_k\}$
- Gegeben Menge  $\mathbf{X} \times \mathbf{y} \subset \mathcal{X} \times \mathcal{Y}$  bei der jedem Beispiel  $\mathbf{x}_i$  die zugehörige Klasse zugeordnet ist:  $(\mathbf{x}_i, y_i)$
- Qualitätsfunktion  $q : (\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \rightarrow \mathcal{Y}) \rightarrow \mathbb{R}$

### Ziel:

- Finde Modell

$$f : \mathcal{X} \rightarrow \mathcal{Y},$$

das die Qualitätsfunktion optimiert.

**Lernen als Optimierungsproblem!**

## Beispiel: Klassifikation von Schwertlilien

- Klassen:  $\mathcal{Y} = \{\text{setosa}, \text{versicolor}, \text{virginica}\}$
- Menge  $\mathbf{X} \times \mathbf{y}$  mit 150 Beispiele mit Spalte "species"
- Qualitätsfunktion

$$q(\mathbf{X} \times \mathbf{y}, f) = \sum_{(x, y) \in \mathbf{X} \times \mathbf{y}} \underbrace{\text{err}(y, f(x))}_{=\hat{y}}, \quad \text{err}(y, \hat{y}) = \begin{cases} 0, & \text{falls } y = \hat{y} \\ 1, & \text{sonst.} \end{cases}$$

**Beispiel: Klassifikation von Schwertlilien**

- Klassen:  $\mathcal{Y} = \{\text{setosa}, \text{versicolor}, \text{virginica}\}$
- Menge  $\mathbf{X} \times \mathbf{y}$  mit 150 Beispiele mit Spalte "species"
- Qualitätsfunktion

$$q(\mathbf{X} \times \mathbf{y}, f) = \sum_{(x, y) \in \mathbf{X} \times \mathbf{y}} \underbrace{\text{err}(y, f(x))}_{=\hat{y}}, \quad \text{err}(y, \hat{y}) = \begin{cases} 0, & \text{falls } y = \hat{y} \\ 1, & \text{sonst.} \end{cases}$$

**Funktion  $q$  zählt die Anzahl der Vorhersagefehler des Modells  $f$  auf der Menge  $\mathbf{X}$**

**Beispiel: Klassifikation von Schwertlilien**

- Klassen:  $\mathcal{Y} = \{\text{setosa}, \text{versicolor}, \text{virginica}\}$
- Menge  $\mathbf{X} \times \mathbf{y}$  mit 150 Beispiele mit Spalte "species"
- Qualitätsfunktion

$$q(\mathbf{X} \times \mathbf{y}, f) = \sum_{(x,y) \in \mathbf{X} \times \mathbf{y}} \underbrace{\text{err}(y, f(x))}_{=\hat{y}}, \quad \text{err}(y, \hat{y}) = \begin{cases} 0, & \text{falls } y = \hat{y} \\ 1, & \text{sonst.} \end{cases}$$

**Funktion  $q$  zählt die Anzahl der Vorhersagefehler des Modells  $f$  auf der Menge  $\mathbf{X}$**

**Ziel:** Finde  $f^*$  mit minimalem  $q(\mathbf{X}, f)$

**Beispiel: Klassifikation von Schwertlilien**

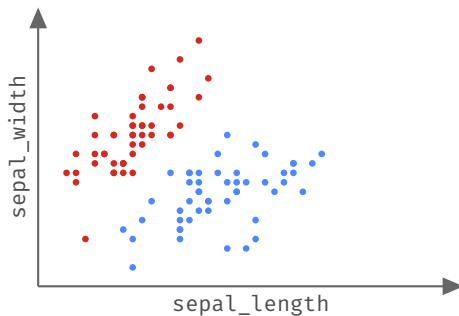
- Klassen:  $\mathcal{Y} = \{\text{setosa}, \text{versicolor}, \text{virginica}\}$
- Menge  $\mathbf{X} \times \mathbf{y}$  mit 150 Beispiele mit Spalte "species"
- Qualitätsfunktion

$$q(\mathbf{X} \times \mathbf{y}, f) = \sum_{(x,y) \in \mathbf{X} \times \mathbf{y}} \underbrace{\text{err}(y, f(x))}_{=\hat{y}}, \quad \text{err}(y, \hat{y}) = \begin{cases} 0, & \text{falls } y = \hat{y} \\ 1, & \text{sonst.} \end{cases}$$

**Funktion  $q$  zählt die Anzahl der Vorhersagefehler des Modells  $f$  auf der Menge  $\mathbf{X}$**

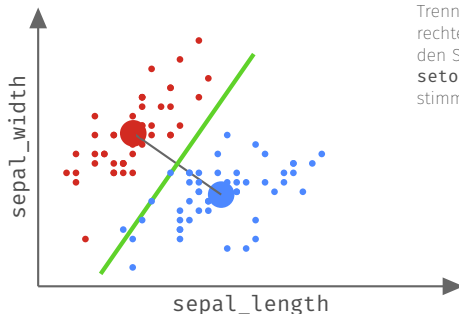
**Ziel:** Finde  $f^*$  mit minimalem  $q(\mathbf{X}, f)$   $\rightarrow$  Optimierungsproblem

## Beispiel: **Klassifikation von Schwertlilien**





## Beispiel: Klassifikation von Schwertlilien

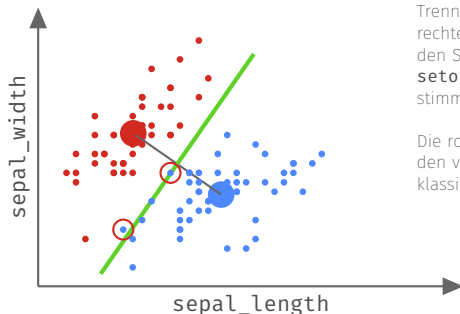


In diesem Fall wurde eine Trenn-Ebene als Mittelsenkrechte auf der Strecke zwischen den Schwerpunkten der Klasse **setosa** und **versicolor** bestimmt.

### Einfacher Algorithmus:

Trenn-Ebene über die Klassenschwerpunkte der Attribute **sepal\_length** und **sepal\_width**

## Beispiel: Klassifikation von Schwertlilien



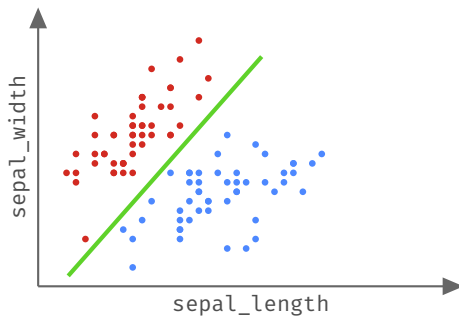
In diesem Fall wurde eine Trenn-Ebene als Mittelsenkrechte auf der Strecke zwischen den Schwerpunkten der Klasse **setosa** und **versicolor** bestimmt.

Die rot umkreisten Punkte werden von der Trenn-Ebene falsch klassifiziert.

### Einfacher Algorithmus:

Trenn-Ebene über die Klassenschwerpunkte der Attribute **sepal\_length** und **sepal\_width**

## Beispiel: Klassifikation von Schwertlilien



Die Daten sind *linear separierbar* – eine andere Ebene schafft dies ohne Fehler.  
Die Optimierung der Qualitätsfunktion sucht nach der besten Ebene.

# Maschinelles Lernen mit Python

## Module für das Maschinelle Lernen

- SciKit-Learn ist umfangreiches Modul mit vielen ML Algorithmen
- SciKit-Learn funktioniert ganz gut mit Pandas
- Keras als Modul für *Deep Learning* mit Python



## Modelle in Python - Welche Funktionen braucht man?

- Anlegen eines neuen, untrainierten Modells
- Anpassen des Modells auf Daten  $\mathbf{X}, \mathbf{y}$
- Vorhersagen  $\hat{\mathbf{y}}$  berechnen auf Daten  $\mathbf{X}'$

## **Beispiel: Modell Trainieren mit SciKit-Learn**

Wir betrachten im Folgenden diese Schritte:

1. Daten lesen
2. Trainings- und Test-Daten aufteilen
3. Modell trainieren
4. Mit Test-Daten das Modell evaluieren

## **Beispiel: Modell Trainieren mit SciKit-Learn**

Wir betrachten im Folgenden diese Schritte:

1. Daten lesen
2. Trainings- und Test-Daten aufteilen
3. Modell trainieren
4. Mit Test-Daten das Modell evaluieren

Die [confusion matrix](#) haben wir auch für mehr als zwei Klassen kennengelernt.

## **Beispiel auf Iris Daten mit allen drei Klassen!**



## Schritt 1: Daten lesen

Das haben wir ja bereits häufiger gemacht (Pandas):

```
import pandas as pd

df = pd.read_csv('data/iris.csv')

# Daraus nehmen wir die Spalten mit Features
# und separat (fuer y) die Spalte mit den Labeln

features = [c for c in df.columns if c != 'species']

X = iris[features]
y = iris['species']
```

## Schritt 2: Trainings- und Test-Daten aufteilen

Eigene Split-Methode war ja Bestandteil von Übungsblatt 4.

Wir schauen uns das jetzt mal mit [SciKit-Learn](#) an:

```
from sklearn.model_selection import train_test_split

# Wir teilen den Datensatz in 80% Trainingsdaten
# und 20% Testdaten auf:
X_tr, X_tst, y_tr, y_tst = train_test_split(X, y,
                                           test_size=0.2)
```

**Hinweis:** Ich habe hier lediglich aus Platzgründen auf der Folie **X\_train** als **X\_tr** und **X\_test** als **X\_tst** abgekürzt.

## Schritt 3: Modell trainieren

Wir erzeugen einen leeren Entscheidungsbaum und trainieren ihn mit den Trainingsdaten:

```
from sklearn.tree import DecisionTreeClassifier

# 'gini' ist der Default Fall als Split-Kriterium,
# wir
# koennten auch 'entropy' nehmen:
model = DecisionTreeClassifier(criterion="gini")

# Den Baum trainieren:
model.fit(X_tr, Y_tr)
```

## Schritt 4: Modell auf Test-Daten ausprobieren

Wir wenden das gelernte Modell mit `predict` auf die Test-Daten an und schauen uns die `confusion matrix` an:

```
from sklearn.metrics import confusion_matrix

# y_hat enthaelt dann die vorhergesagten Werte:
y_predicted = model.predict(X_tst)

# Wir berechnen die Confusion Matrix
# aus (wahrheit, vorhersage):
matrix = confusion_matrix(y_tst, y_predicted)
print(matrix)
```

## Vereinfachte Auswertung: Testfehler

SciKit Learn bietet `accuracy_score` Funktion an, die die relative Vorhersagegüte berechnet:

```
from sklearn.metrics import accuracy_score

y_predicted = model.predict(X_tst)

# 1 - acc_score(..) liefert den relativen Fehler:
err = 1 - accuracy_score(y_predicted, y_test)
```

## Auch interessant - der **classification report**

SciKit-Learn bietet noch einen kleinen Bericht zusätzlich zur **confusion matrix**, nämlich den **classification report**:

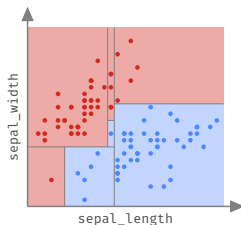
```
from sklearn.metrics import classification_report

# Aus den Test y und den vorhergesagten y_predicted
# kann man den classification report erstellen:
#
report = classification_report(y_tst, y_predicted,
                              digits=3)

print(report)
```

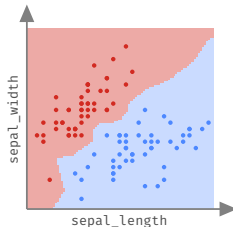
# Klassifikationsverfahren

## Entscheidungsbäume, nächste Nachbarn und lineare Modelle



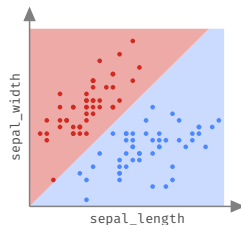
Entscheidungsb Baum

Trennung nach einzelnen  
Attributen, achsenparallel



k-nächste Nachbarn

Trennung in Regionen, nach Di-  
stanz (Berechnung über alle Attribute)

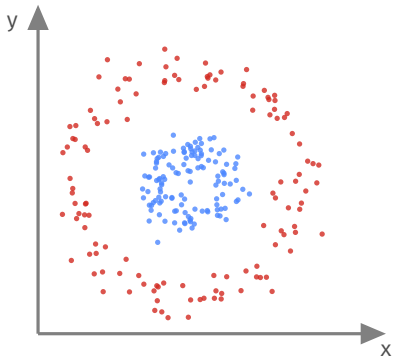


Lineare Modelle

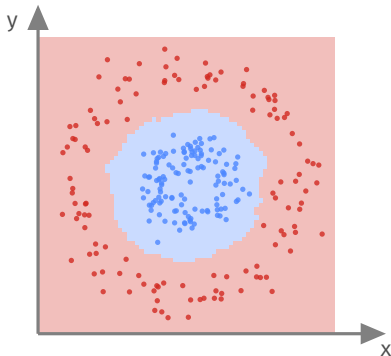
Trennung mit linearer  
Funktion über alle Attribute



**Betrachten wir einen anderen Datensatz:**



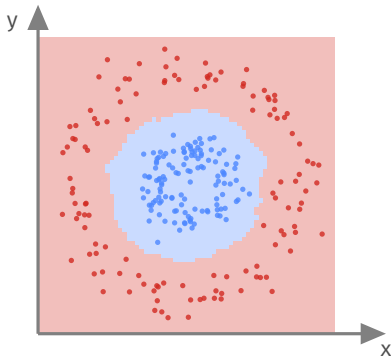
**Betrachten wir einen anderen Datensatz:**



Mit  $k$ -NN Modell kein Problem.

**$k$ -NN ist aber sehr langsam bei der Vorhersage! :(**

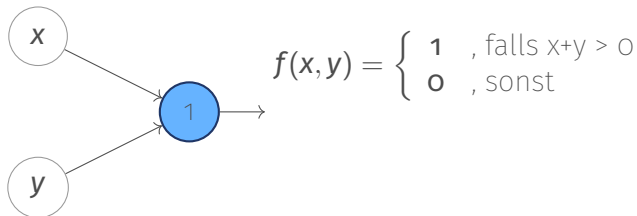
**Betrachten wir einen anderen Datensatz:**



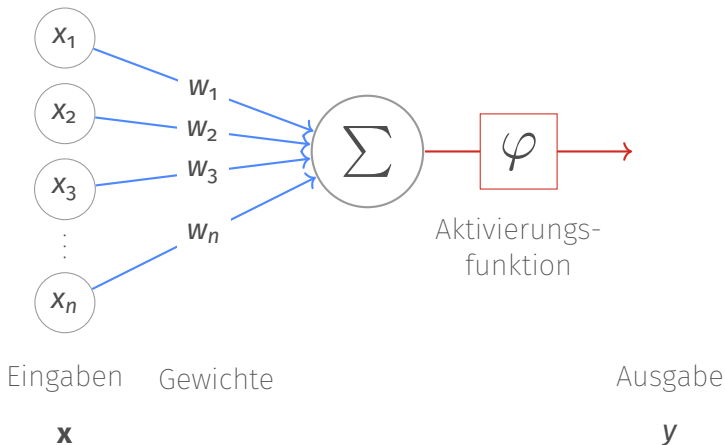
Mit **linearem** Modell nicht lösbar?

# Neuronale Netze (und ChatGPT)

- Modellierung nach Neuronen (Gehirn)
- kleine Zellen die Input zu Output verarbeiten
- Basiert auf der Idee des Perzeptrons (Rosenblatt, 1957)



## Neuron in einem Neuronalen Netz



## Aktivierungsfunktion

Berechnet Ausgabe aus gewichteter Eingabe-Summe.  
Wie von X und Y in [?] geschrieben

Identität

$$\text{id}(\mathbf{x}) = \mathbf{x}$$



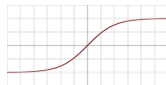
Binärer Step

$$\varphi(\mathbf{x}) = \begin{cases} 0, & \text{falls } \mathbf{x} < 0 \\ 1, & \text{sonst} \end{cases}$$



Tangens  
Hyperbolicus

$$\tanh(\mathbf{x}) = \frac{e^{2\mathbf{x}} - 1}{e^{2\mathbf{x}} + 1}$$



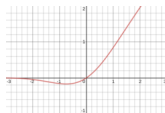
## Weitere Aktivierungsfunktionen

ReLU<sup>1</sup>

$$(x)^+ = \begin{cases} 0, & \text{falls } x \leq 0 \\ x, & x > 0 \end{cases}$$

GELU<sup>2</sup>

$$g(x) = x \frac{1}{2} [1 + \text{erf}(x/\sqrt{2})]$$



[1] *Rectified linear units improve restricted boltzmann machines*, Nair, Vinod and Hinton, Geoffrey E., Proceedings of the 27th International Conference on International Conference on Machine Learning, 2010

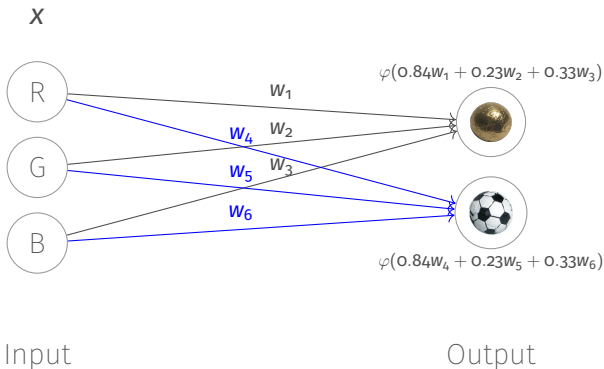
[2] *Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units*, Dan Hendrycks and Kevin Gimpel, Computing Research Repository (CoRR), 2016



## Funktionen mit neuronalen Netzen



R = 0,84  
G = 0,23  
B = 0,33

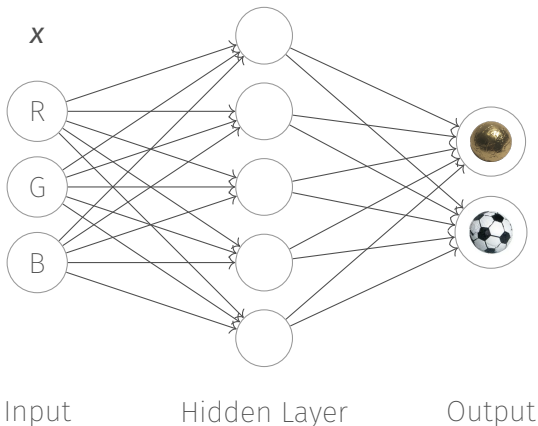


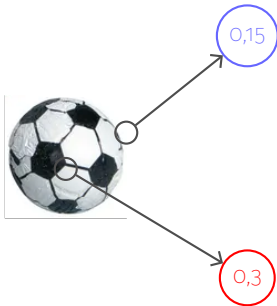
Ziel: Finde die Gewichte  $w_i$ , die zur genauesten Vorhersage führen!

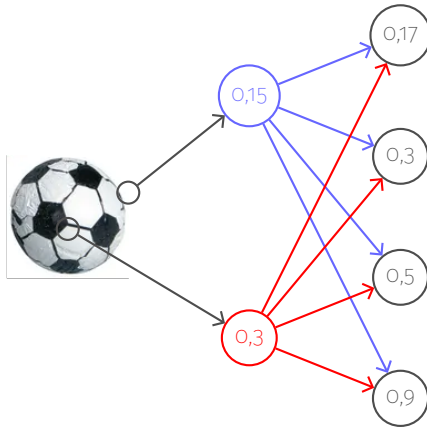
## Komplexe Funktionen mit neuronalen Netzen

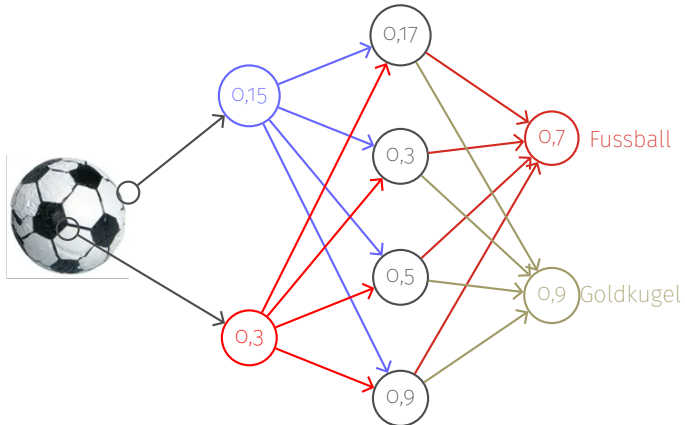


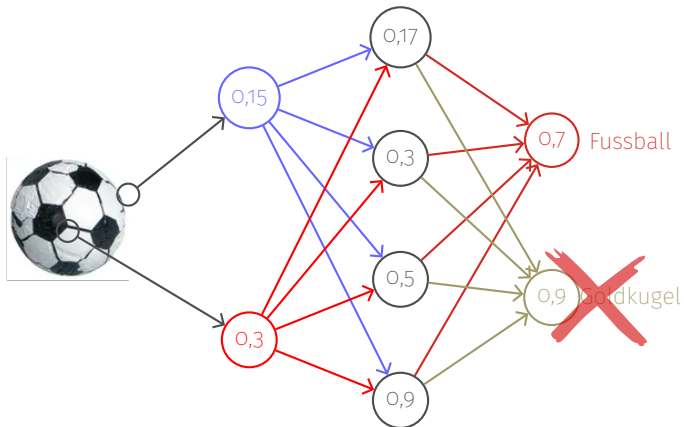
R = 84%  
G = 23%  
B = 33%













0,15

0,3

0,17

0,3

0,5

0,9

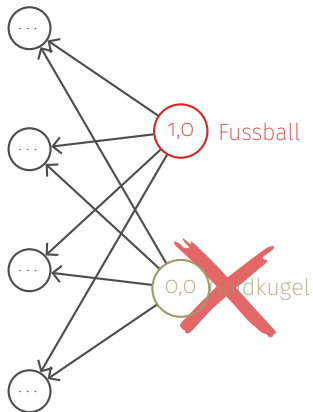
1,0 Fussball

~~0,0 Handkugel~~

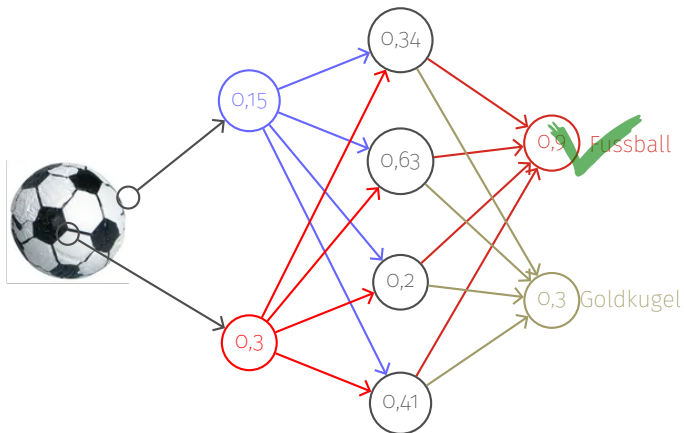


0,15

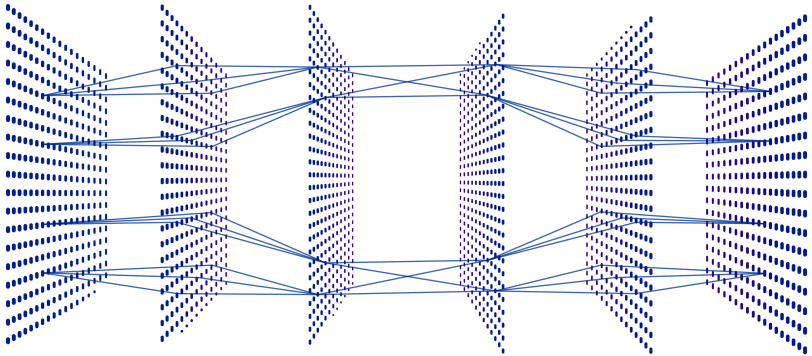
0,3







## Deep Learning



## ChatGPT

- Generative AI, entwickelt von OpenAI
- Tiefes, Neuronales Netz (viele Layer/Ebenen)
- Verschiedene Ebenen, die unterschiedliche Aufgaben ausführen
- 175 Milliarden Parameter

## ChatGPT

- Generative AI, entwickelt von OpenAI
- Tiefes, Neuronales Netz (viele Layer/Ebene)
- Verschiedene Ebenen, die unterschiedliche Aufgaben ausführen
- 175 Milliarden Parameter
- Klassifikation mit Textbeispielen zum Lernen

**Beispiel:** "Das Konzert war großartig."

**Klasse:** "positive Stimmung"

## ChatGPT

- Generative AI, entwickelt von OpenAI
- Tiefes, Neuronales Netz (viele Layer/Ebene)
- Verschiedene Ebenen, die unterschiedliche Aufgaben ausführen
- 175 Milliarden Parameter
- Klassifikation mit Textbeispielen zum Lernen
  - Beispiel:** "Das Konzert war großartig."
  - Klasse:** "positive Stimmung"
- Bestärkendes Lernen aus Interaktion
  - Benutzer:** "Deine vorherige Antwort war hilfreich."
  - Verstärkung:** Positives Signal, um die Modellgewichtungen anzupassen.

# Model Selection

## Entscheidungsbäume

- + leicht zu interpretieren
- + schnell in der Vorhersage
- Nur jeweils ein Attribut zum verzweigen
- neigt zu Overfitting
- nur Achsen-parallele Entscheidungslinien

## **$k$ -Nearest-Neighbors**

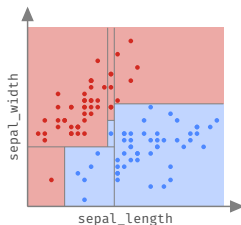
- + leicht verständlich
- + robustes Verfahren, wenig Overfitting (für größeres  $k$ )
- u.U. sehr langsam in der Vorhersage
- Distanzfunktion kann schwierig werden (Welche? Problemspezifisch?)
- Problematisch in hohen Dimensionen (spärliche Datenbereiche)



## **SVM (linear/nicht-linear)**

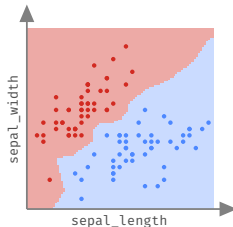
- + schnell in der Vorhersage
- + Funktioniert auch in hohen Dimensionen gut (Texte)
- Interpretierbarkeit?
- Auswahl der Kern-Funktion?
- Einstellungen für Parameter?

## Entscheidungsbäume, nächste Nachbarn und lineare Modelle



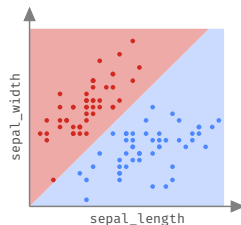
Entscheidungsb Baum

Trennung nach einzelnen  
Attributen, achsenparallel



k-nächste Nachbarn

Trennung in Regionen, nach Di-  
stanz (Berechnung über alle Attribute)

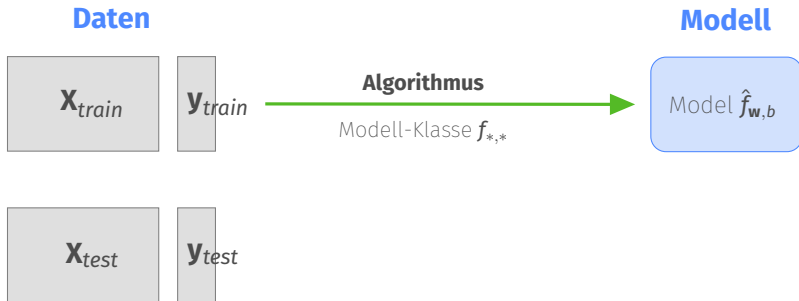


Lineare Modelle

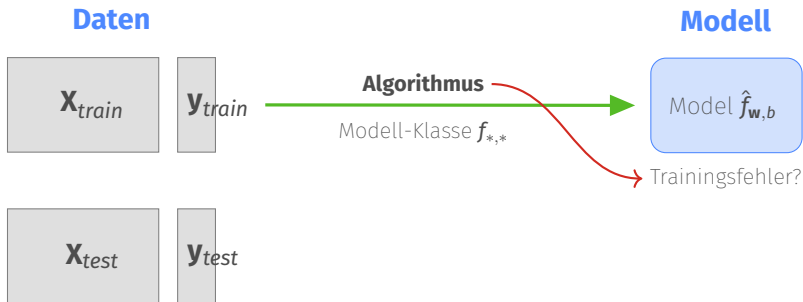
Trennung mit linearer  
Funktion über alle Attribute

### Welches Verfahren für meine Daten?

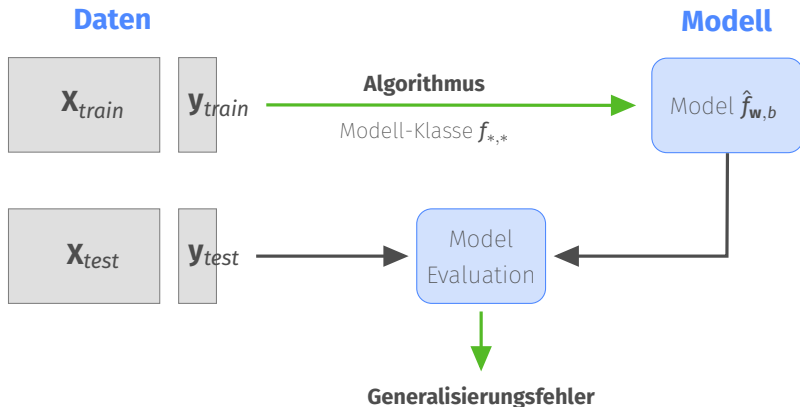
## Vorgehen beim überwachten Lernen



## Vorgehen beim überwachten Lernen



## Vorgehen beim überwachten Lernen



## Algorithmus-Auswahl:

- Wählt den Lern-Algorithmus, der das Modell auswählen soll
- Jeder Algorithmus ist auf Modell-Klasse beschränkt

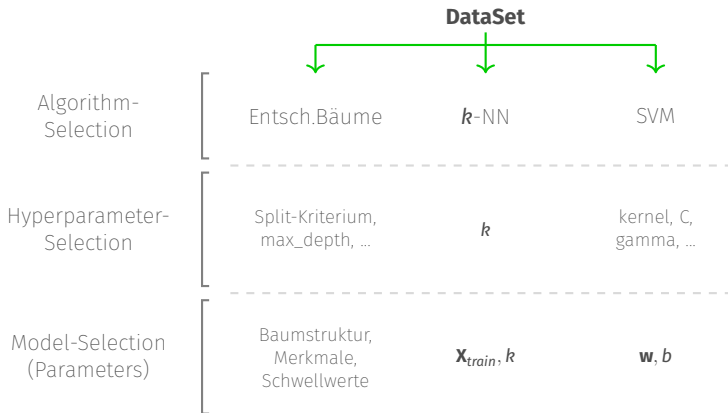
## Hyperparameter-Auswahl:

- Wählt die Parameter für den Lern-Algorithmus
- Parameter bestimmen, wie das beste Modell gesucht wird

## Parameter-Auswahl:

- Auswahl der Modell-Parameter durch den Algorithmus
- Modell-Parameter legen *ein* Modell aus Modell-Klasse fest

## Suche nach dem besten Modell



# Organisatorisches / Wie geht's weiter?

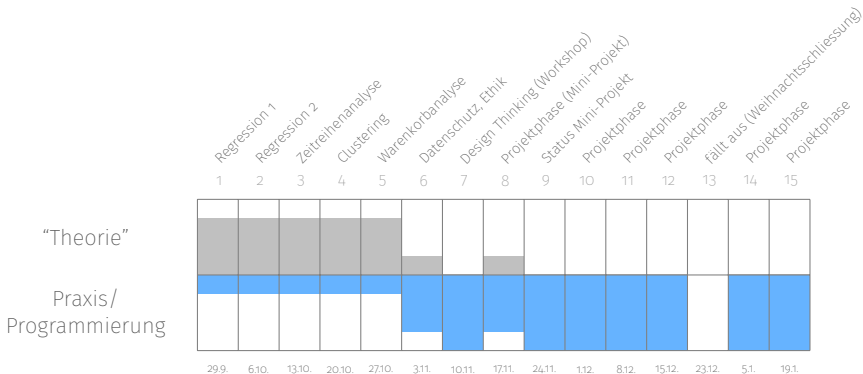


## Prüfungsleistung - Hausarbeit

- Prüfungsleistung ist Hausarbeit
- Aufgabenstellung wird am 11.7.2024 vorgestellt
- Bearbeitung bis 11.8.2024 um 23:59 Uhr  
(Abgabe: PDF des Notebooks in Moodle hochladen)
  
- Gruppenarbeit mit max. 3 Personen möglich
- Selbstorganisation der Gruppen, bitte bis 12.7.2024  
Gruppeneinteilung verbindlich mitteilen (Mail)

# Was erwartet Sie in Data Science 2?

## Aufbau der Vorlesung



## Ziel des Kurses

- Datengetriebenes Denken fördern
- Lern-Probleme in Anwendungen identifizieren
- Ideen für Data Science Lösungen entwickeln
- Exploration+Prototyping von Daten/Modellen

## Projektphase

- Zusammenhängenden Anwendungsfall bearbeiten
- Unterschiedliche Aufgaben in gleichem Fallbeispiel
- Gruppenarbeit, gemischte Studiengänge (!?)

## Prüfungsleistung

- Präsentation des Projektes (Vortrag)
- Hausarbeit über das Projekt ca. 6-8 Seiten im LNCS-Stil
- Bewertung von 2 Hausarbeiten anderer Gruppen (Review)
- Abgabe + Präsentation in Gruppen