Data Science 1

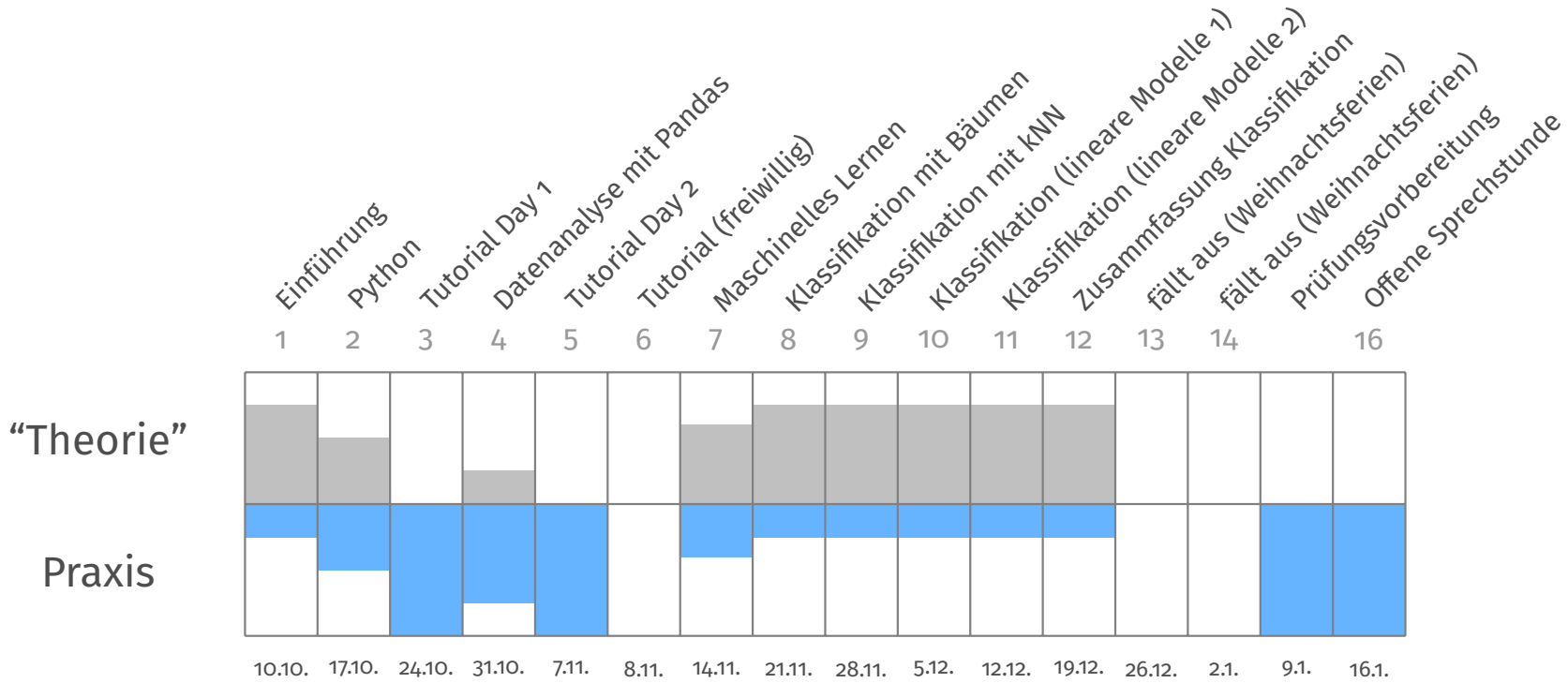
Vorlesung 9

Prof. Dr. Christian Bockermann

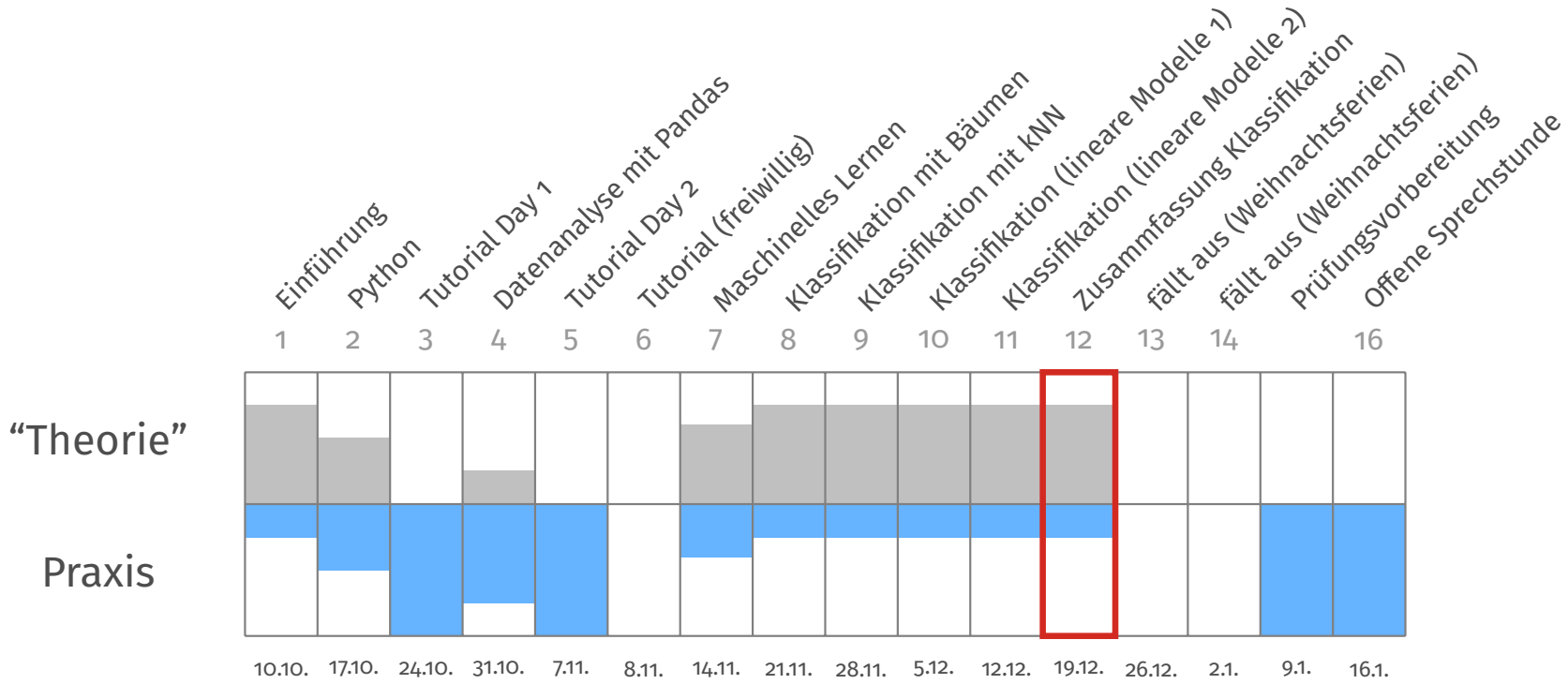
Hochschule Bochum

Wintersemester 2023/2024

Wo sind wir heute?



Wo sind wir heute?



Inhalt

Zusammenfassung Klassifikation

Lern-Algorithmen erwarten Daten häufig in Form einer Tabelle:

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

Beispiel $\mathbf{x}_2 = (x_{a_1}, x_{a_2}, \ldots, x_{a_4}, y)$
$= (0, 1, \ldots, 1, \mathbf{+1})$

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

a_1	a_2	$\cdots$	a_d	y
0	0	$\cdots$	1	**-1**
0	1	$\cdots$	1	**+1**
1	0	$\cdots$	1	**-1**

Trainingsdaten **X**, **y**

Algorithmus/
Optimierung

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

Modell

ML - Modell Training

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

Trainingsdaten **X**, **y**

Algorithmus/
Optimierung

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

Modell

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

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

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

Trainingsdaten **X**, **y**

Algorithmus/
Optimierung

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

Modell

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

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

Vorhersage

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

Klassifikation ordnet Beispielen diskreten Klassen zu

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

Ziel:

- Finde Modell

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

das die Qualitätsfunktion optimiert.

Klassifikation ordnet Beispielen diskreten Klassen zu

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

Ziel:

- Finde Modell

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

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

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

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

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

Funktion q zählt die Anzahl der Vorhersagefehler des Modells f auf der Menge 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

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

Funktion q zählt die Anzahl der Vorhersagefehler des Modells f auf der Menge X

Ziel: Finde f^* mit minimalem $q(X, f)$

Beispiel: Klassifikation von Schwertlilien

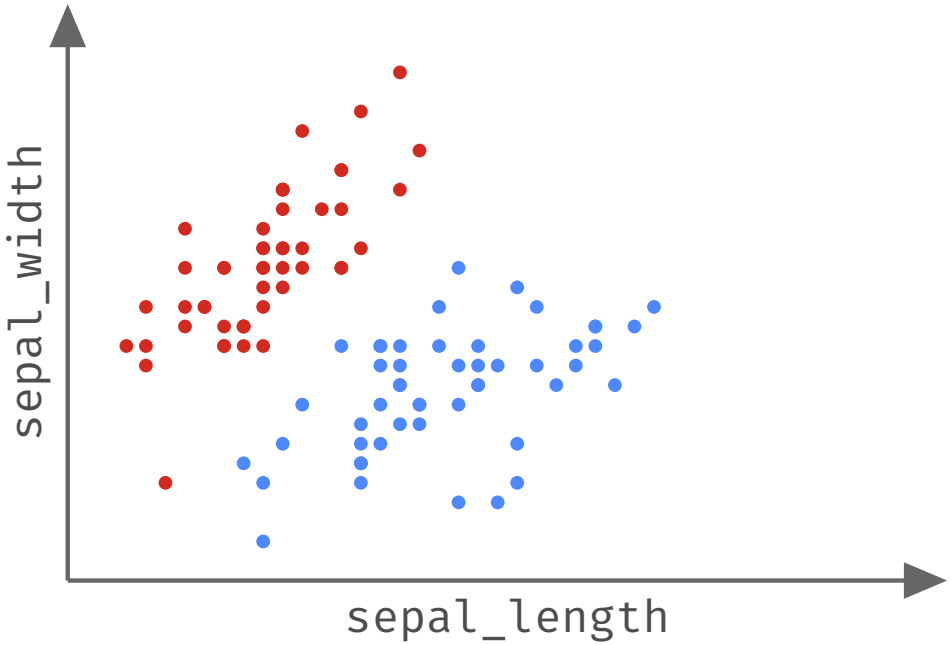
- Klassen: $\mathcal{Y} = \{\text{setosa}, \text{versicolor}, \text{virginica}\}$
- Menge $\mathbf{X} \times \mathbf{y}$ mit 150 Beispiele mit Spalte "`species`"
- Qualitätsfunktion

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

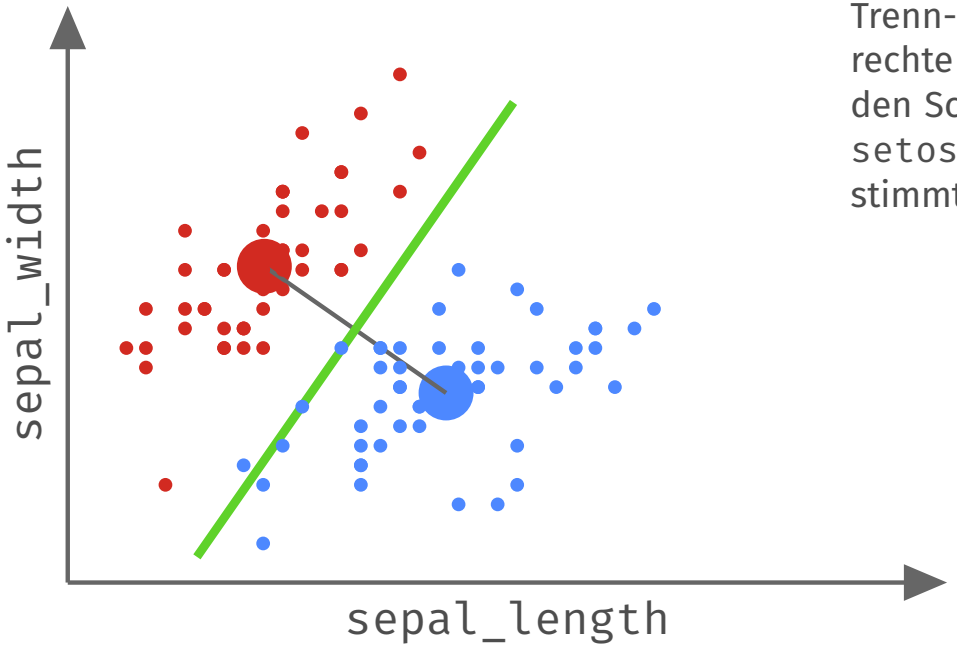
Funktion q zählt die Anzahl der Vorhersagefehler des Modells f auf der Menge X

Ziel: Finde f^* mit minimalem $q(X, f)$ ⟶ Optimierungsproblem

Beispiel: Klassifikation von Schwertlilien



Beispiel: Klassifikation von Schwertlilien

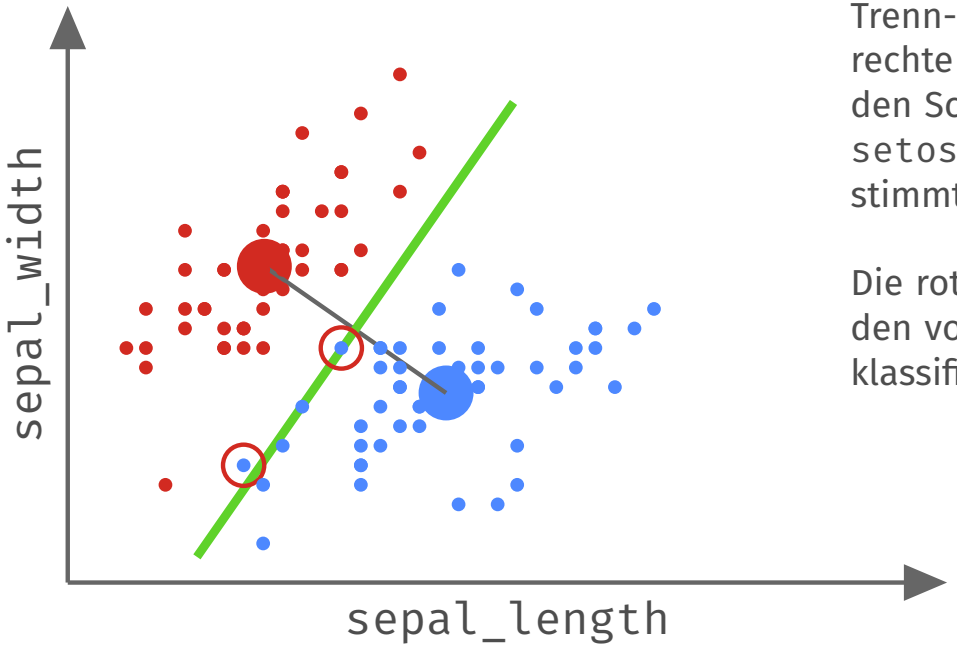


In diesem Fall wurde eine Trenn-Ebene als Mittelsenkrechte auf der Strecke zwischen den Schwerpunkten der Klasse `setosa` und `versicolor` bestimmt.

Einfacher Algorithmus:

Trenn-Ebene über die Klassenschwerpunkte der Attribute `sepal_length` und `sepal_width`

Beispiel: Klassifikation von Schwertlilien



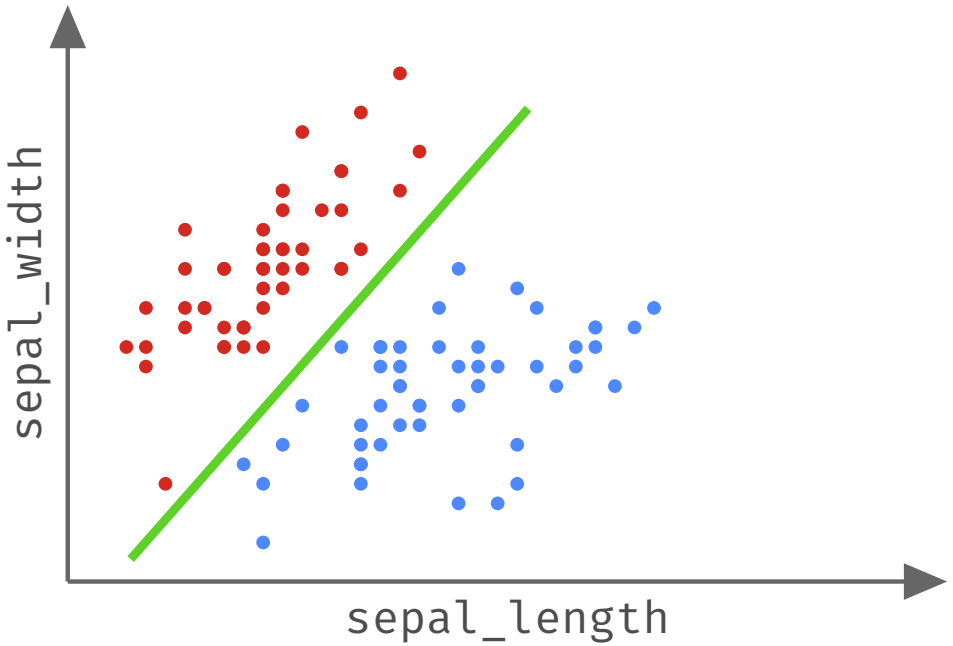
In diesem Fall wurde eine Trenn-Ebene als Mittelsenkrechte auf der Strecke zwischen den Schwerpunkten der Klasse `setosa` und `versicolor` bestimmt.

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 trainineren 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

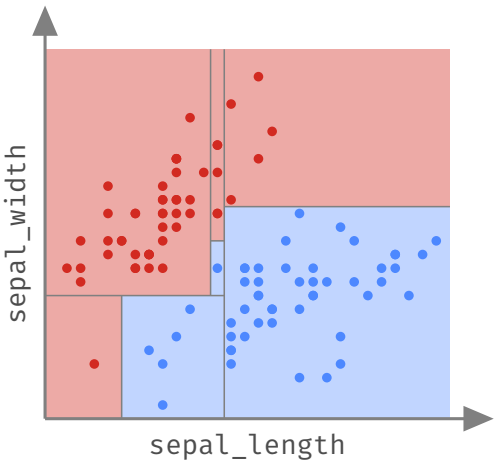
SciKit-Learn bietet noch einen kleinen Bericht zusätzlich zur confusion matrix, nämlich den **classification report**:

```
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)
```

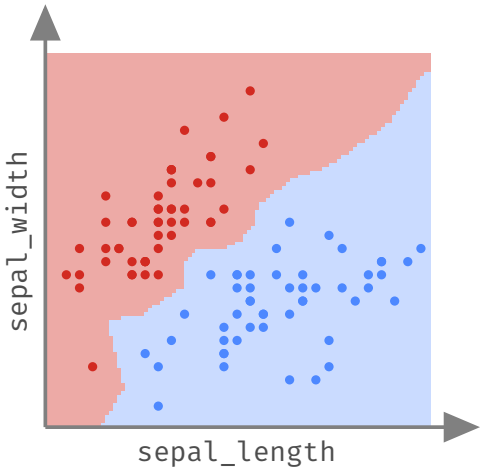
Klassifikationsverfahren

Entscheidungsbäume, nächste Nachbarn und lineare Modelle



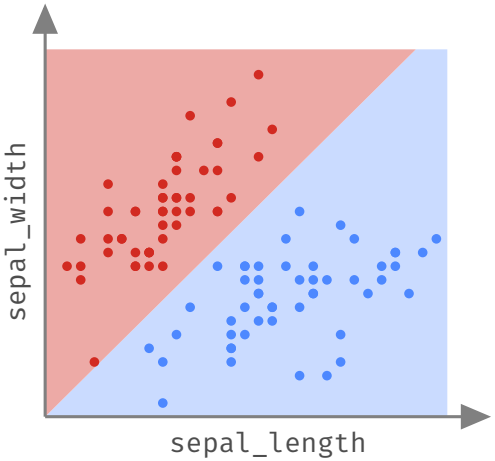
Entscheidungsbaum

Trennung nach einzelnen Attributen, achsenparallel



k-nächste Nachbarn

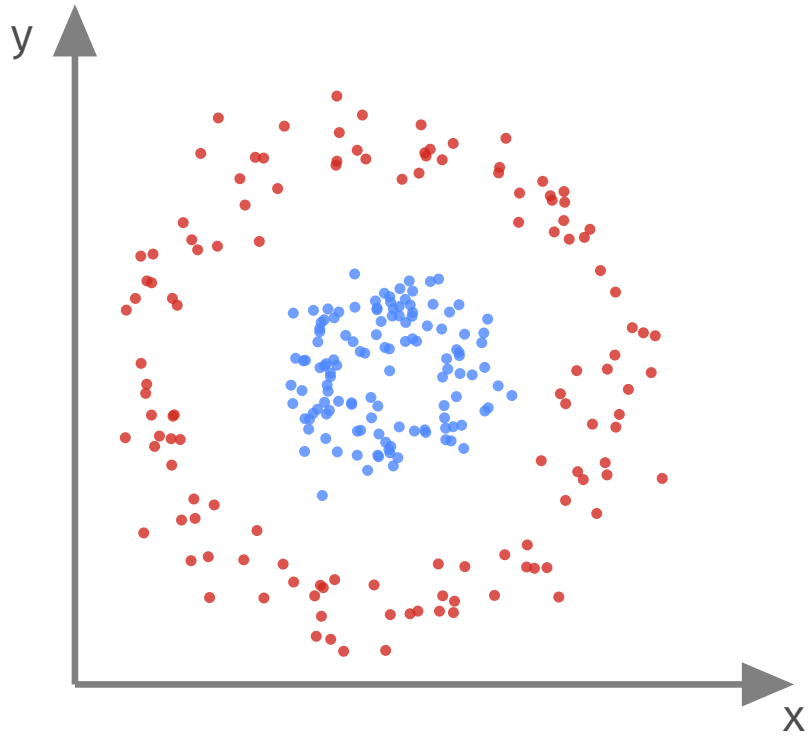
Trennung in Regionen, nach Distanz (Berechnung über alle Attribute)



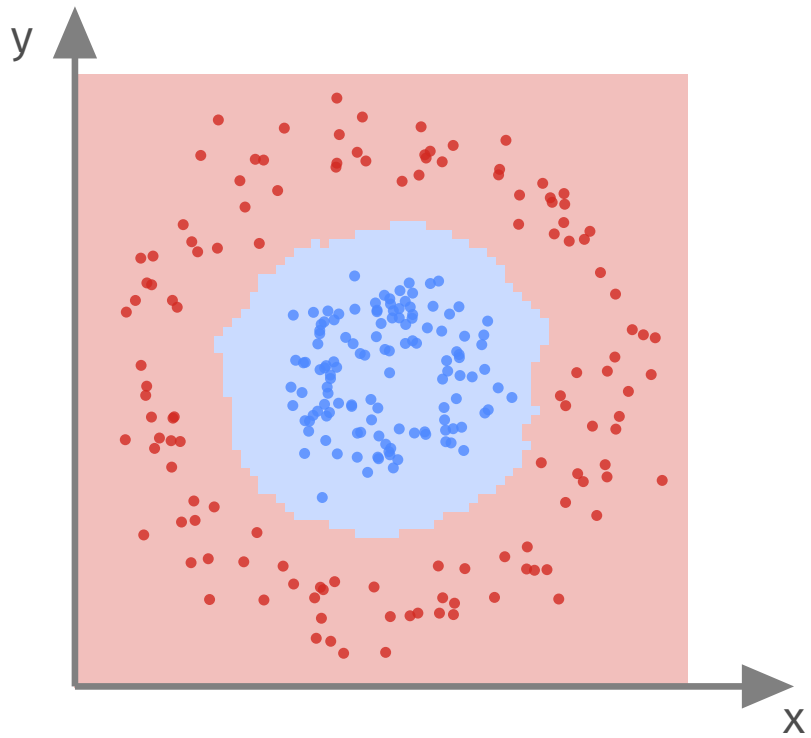
Lineare Modelle

Trennung mit linearer Funktion über alle Attribute

Betrachten wir einen anderen Datensatz:



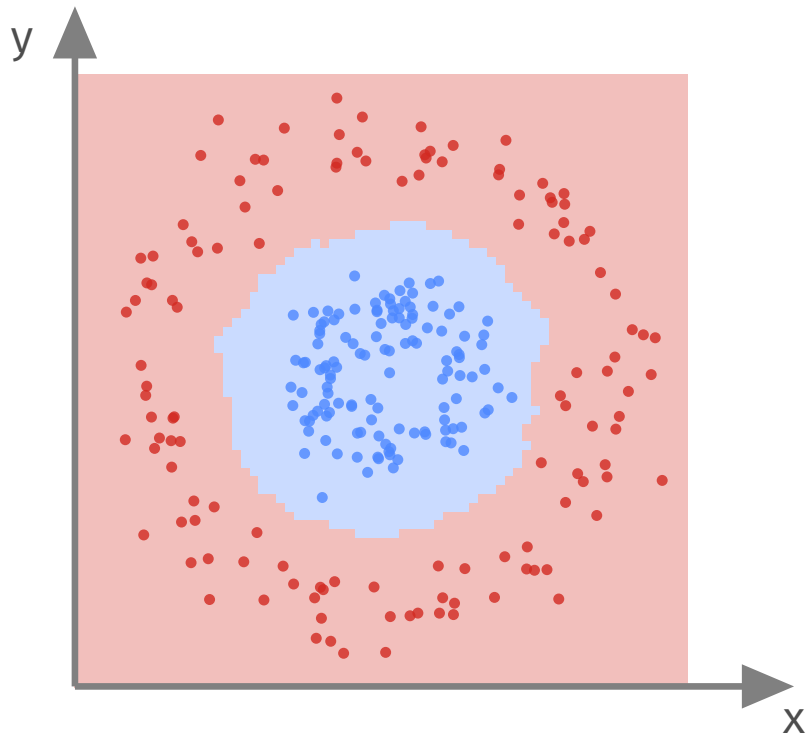
Betrachten wir einen anderen Datensatz:



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?

Model Selection

Entscheidungsbäume

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

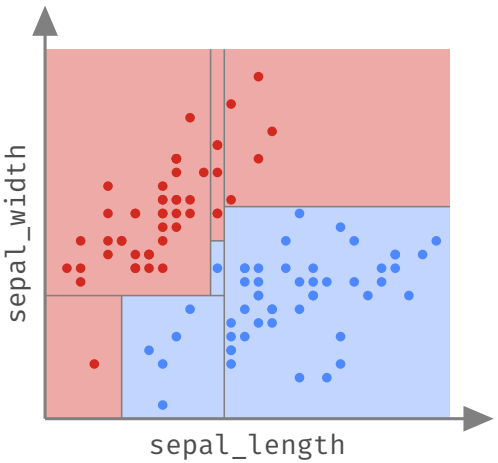
k-Nearest-Neighbors

+ 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)

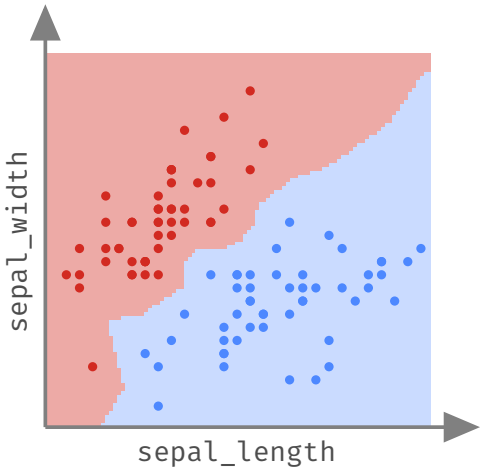
+ schnell in der Vorhersage
+ Funktioniert auch in hohen Dimensionen gut (Texte)
- Interpretierbarkeit?
- Auswahl der Kern-Funktion?
- Einstellungen für Parameter?

Entscheidungsbäume, nächste Nachbarn und lineare Modelle



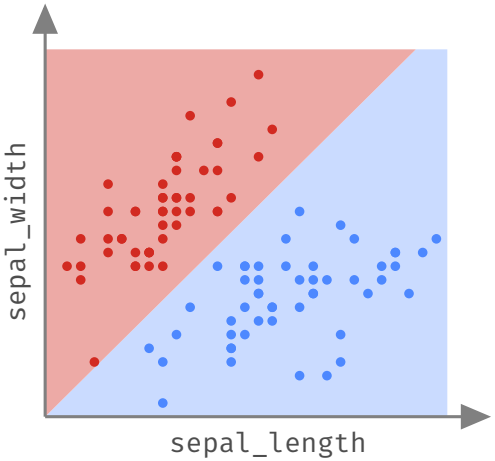
Entscheidungsbaum

Trennung nach einzelnen Attributen, achsenparallel



k-nächste Nachbarn

Trennung in Regionen, nach Distanz (Berechnung über alle Attribute)

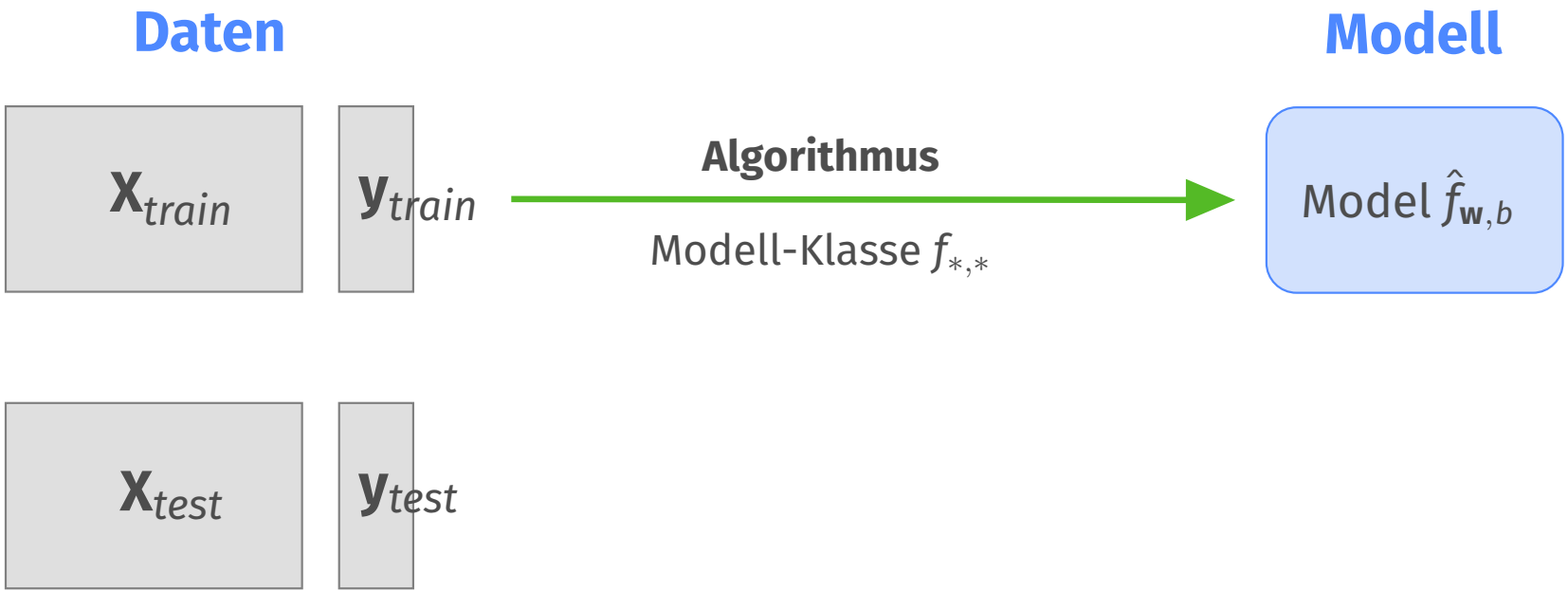


Lineare Modelle

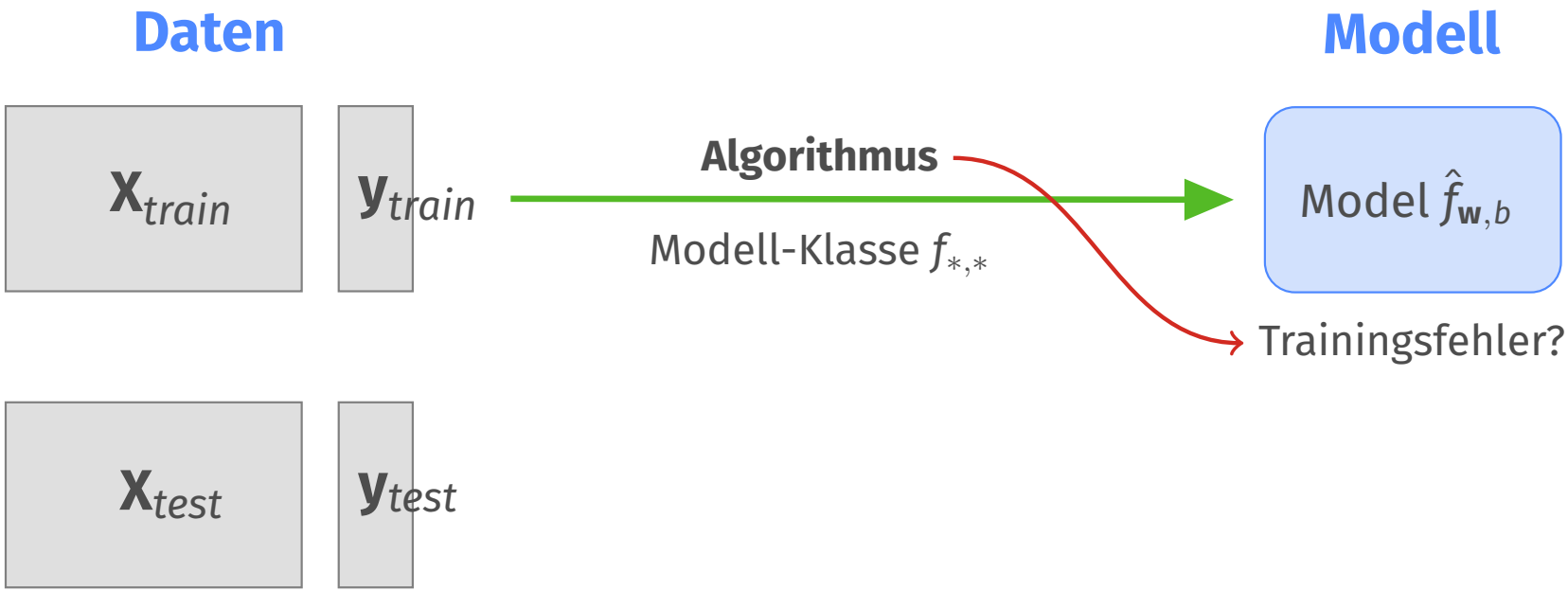
Trennung mit linearer Funktion über alle Attribute

Welches Verfahren für meine Daten?

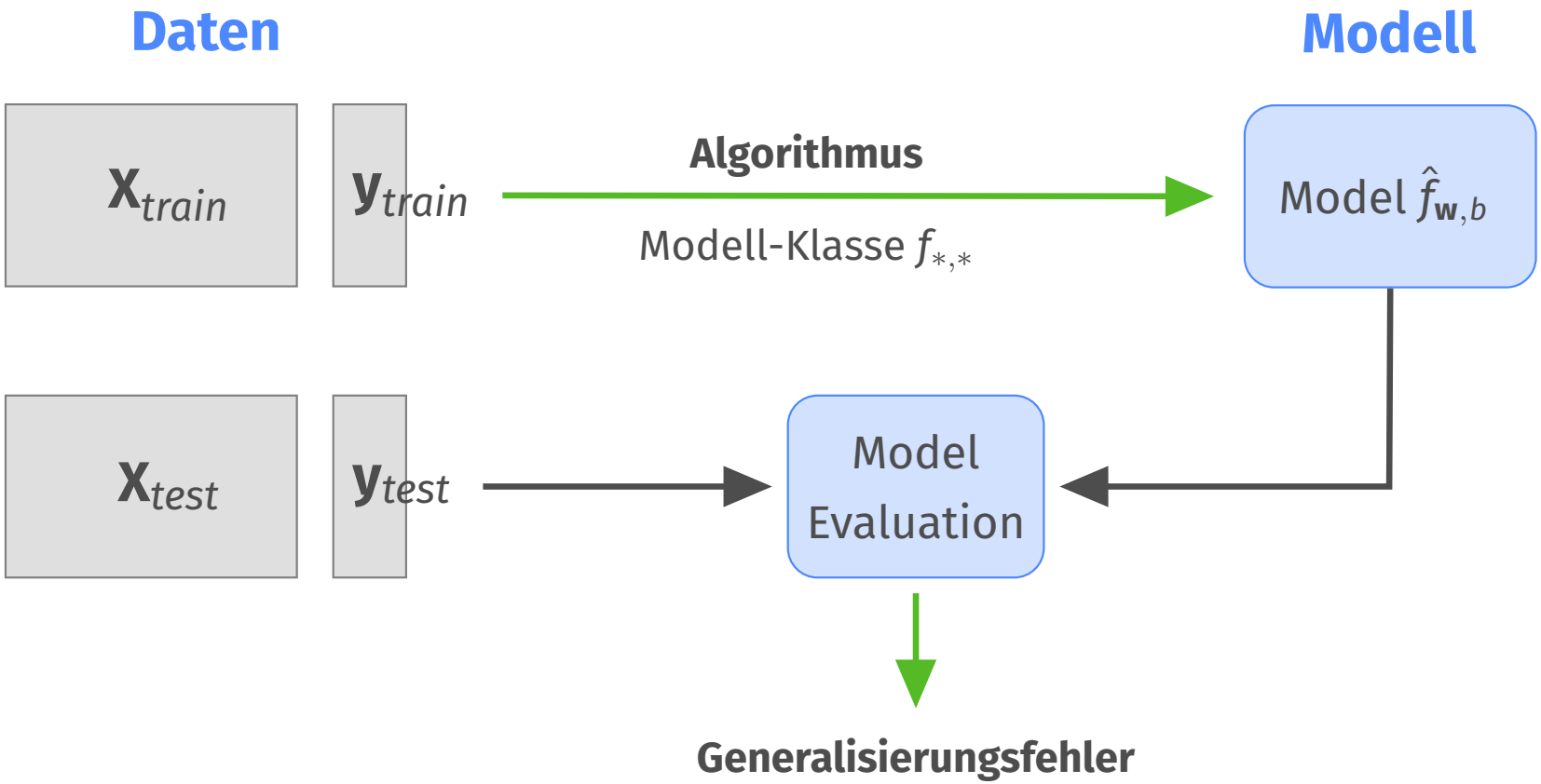
Vorgehen beim überwachten Lernen



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

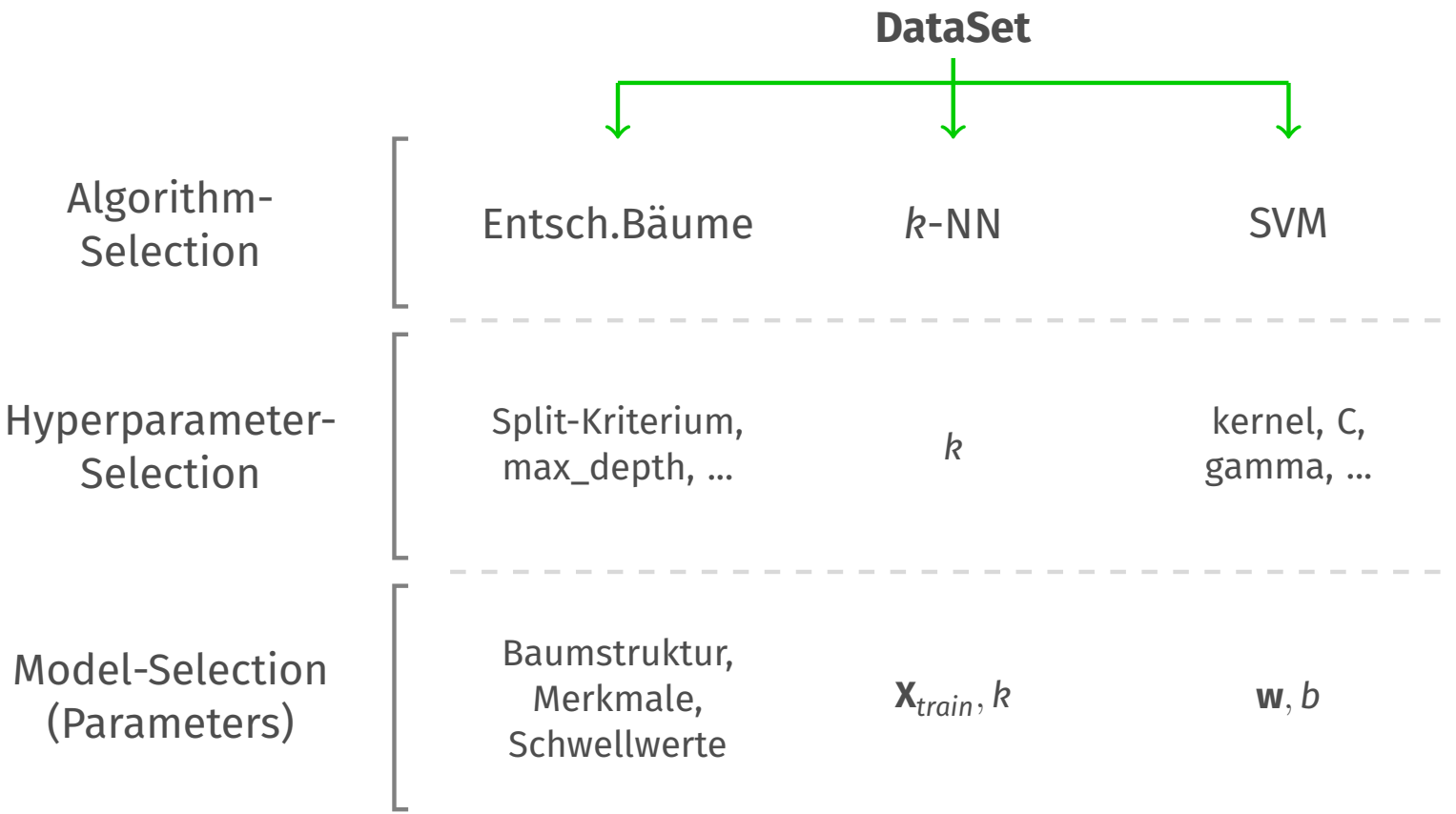
Hyperparameter-Auswahl:

- 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?

Lehre Evaluation

Zur Evaluation der Vorlesung Data Science 1:

https://hsbo.de/ds1ev

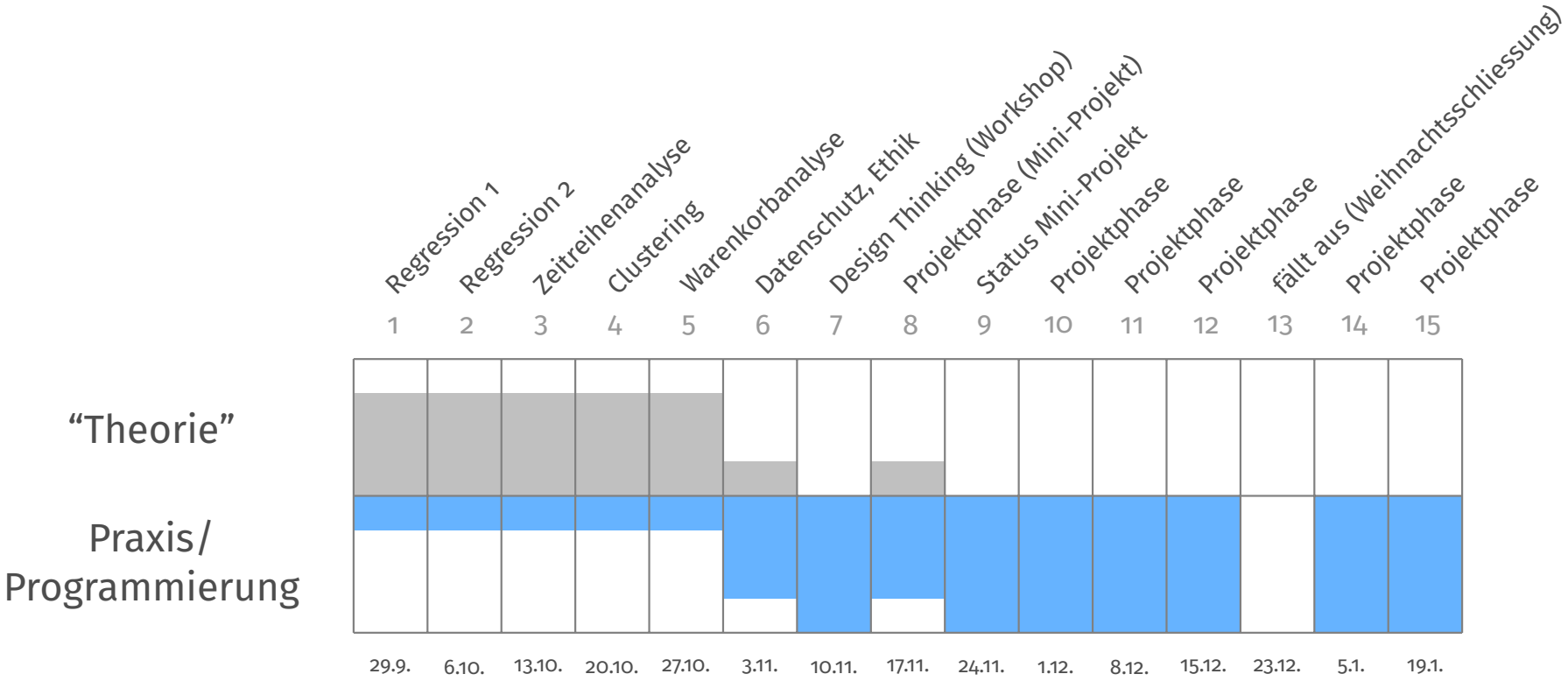
Prüfungsleistung - Hausarbeit

- Prüfungsleistung ist Hausarbeit
- Aufgabenstellung wird am 9.1.2024 vorgestellt
- Bearbeitung bis 18.2.2023 um 23:59 Uhr
  (Abgabe: PDF des Notebooks in Moodle hochladen)

- Gruppenarbeit mit max. 3 Personen möglich
- Selbstorganisation der Gruppen, bitte bis 10.2.2024 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. 8-10 Seiten
- Abgabe + Präsentation in Gruppen