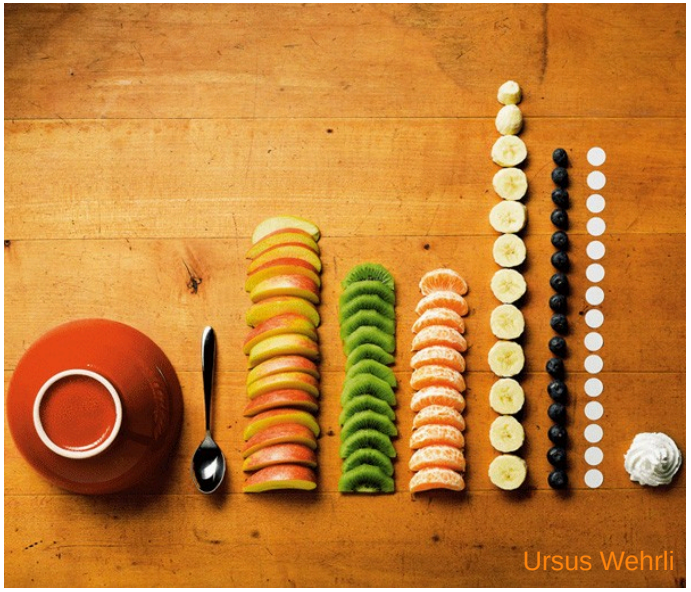


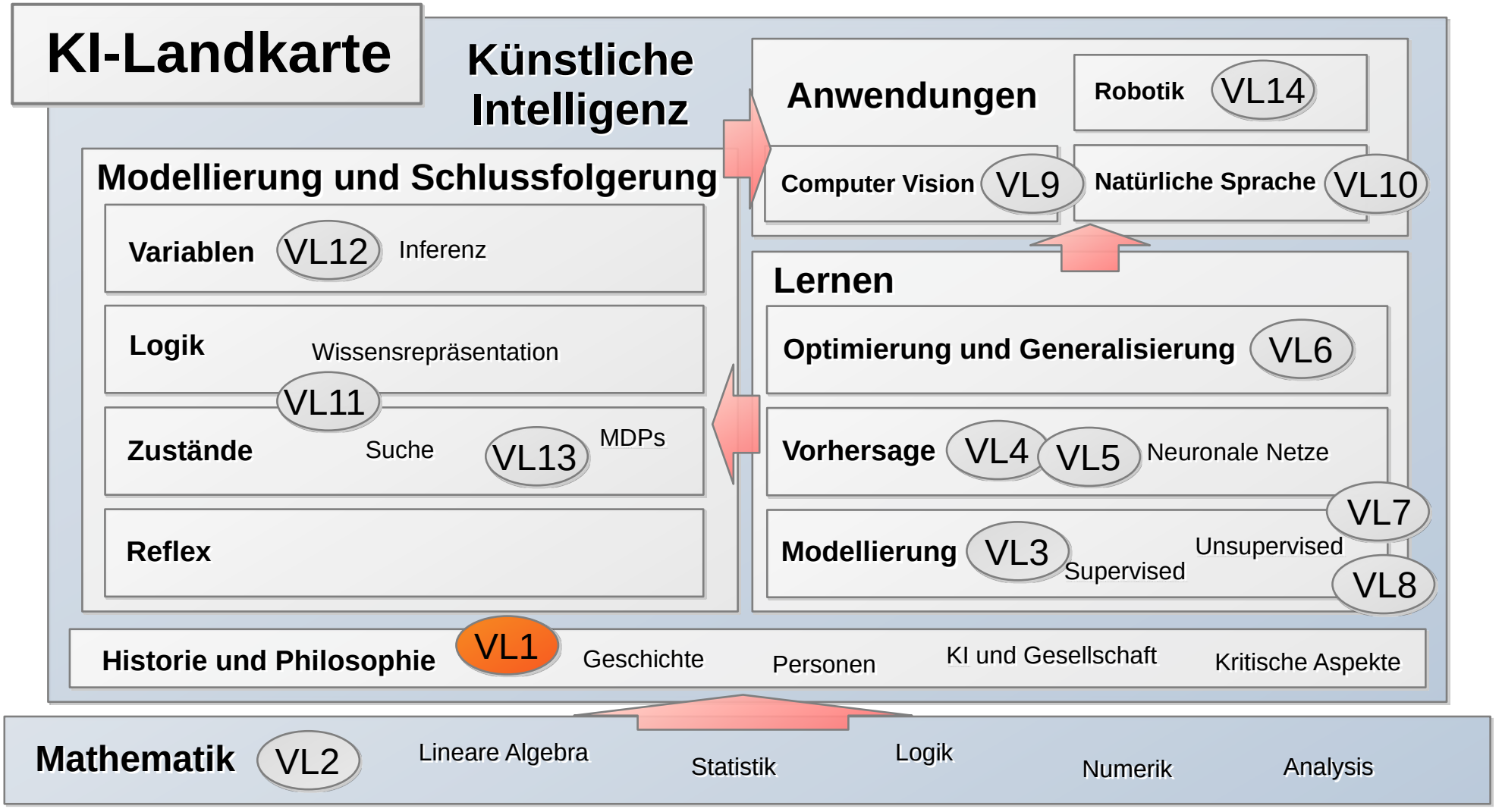
Grundlagen der Künstlichen Intelligenz

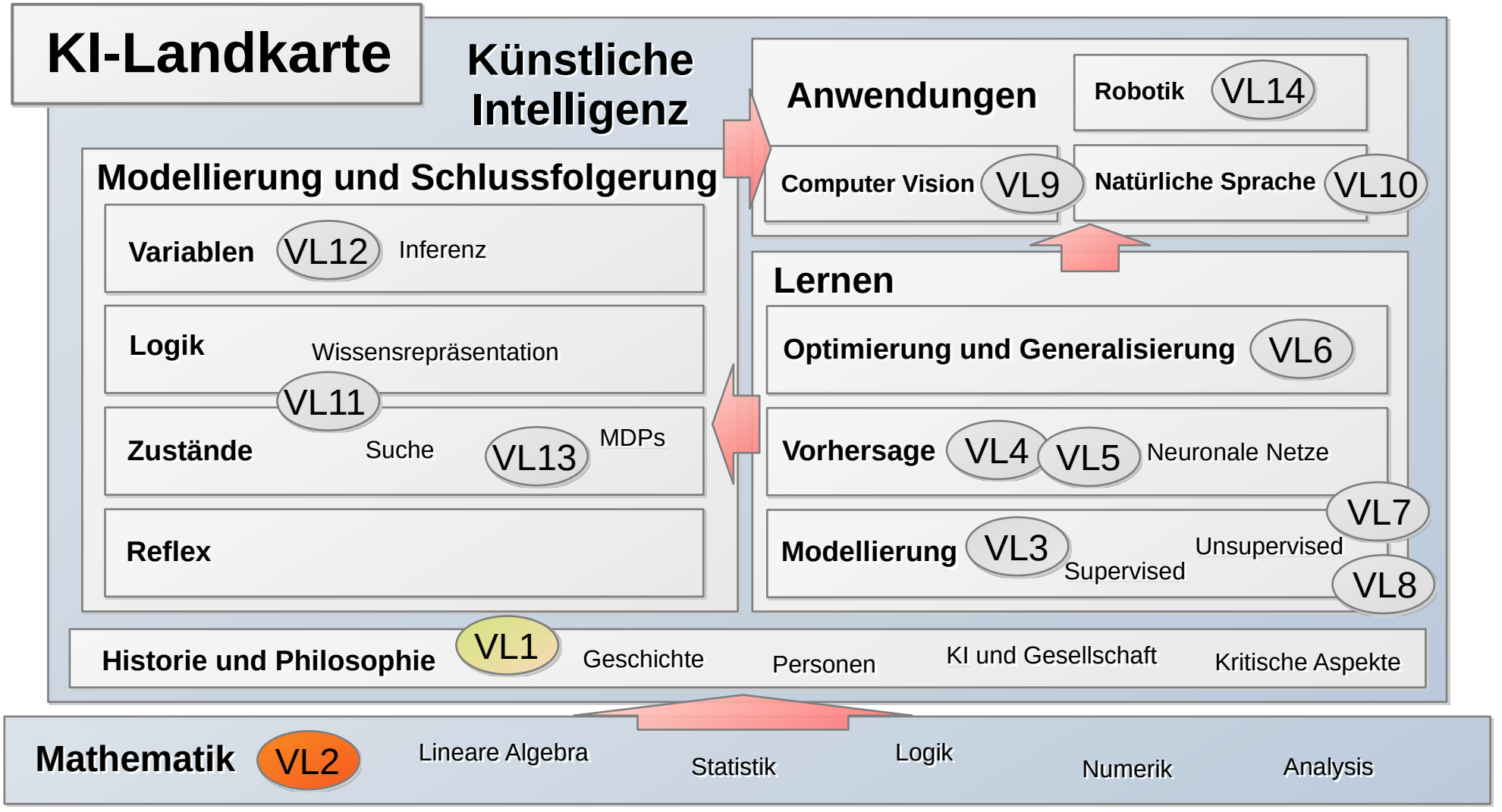
Wintersemester 25/26

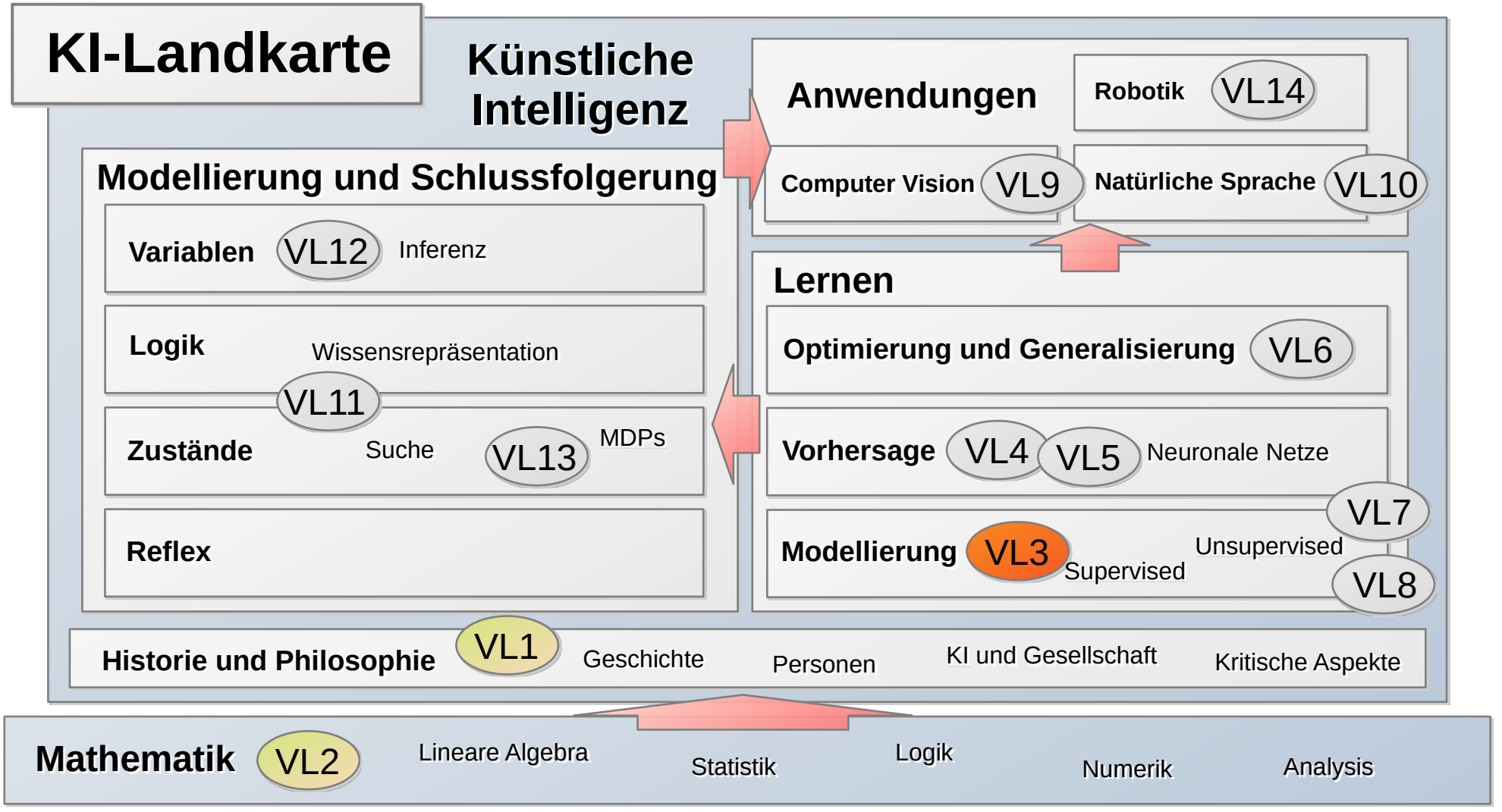
Vorlesung 3

Maschinelles Lernen I:
Lernende Algorithmen,
Klassifikation & Logistische Regression

Prof. Dr. Pascal Friederich
T.T.-Prof. Dr. Peer Nowack


KI-Landkarte
Künstliche Intelligenz
Anwendungen
Robotik
VL14
Computer Vision
VL9
Natürliche Sprache
VL10
Modellierung und Schlussfolgerung
Variablen
VL12
Inferenz
Logik
Wissensrepräsentation
VL11
Zustände
Suche
VL13
MDPs
Reflex
Lernen
Optimierung und Generalisierung
VL6
Vorhersage
VL4
VL5
Neuronale Netze
Modellierung
VL3
Supervised
Unsupervised
VL7
VL8
Historie und Philosophie
VL1
Geschichte
Personen
KI und Gesellschaft
Kritische Aspekte
Mathematik
VL2
Lineare Algebra
Statistik
Logik
Numerik
Analysis


KI-Landkarte
Künstliche Intelligenz
Anwendungen
Robotik
VL14
Computer Vision
VL9
Natürliche Sprache
VL10
Modellierung und Schlussfolgerung
Variablen
VL12
Inferenz
Logik
Wissensrepräsentation
VL11
Zustände
Suche
VL13
MDPs
Reflex
Lernen
Optimierung und Generalisierung
VL6
Vorhersage
VL4
VL5
Neuronale Netze
VL7
Modellierung
VL3
Supervised
Unsupervised
VL8
Historie und Philosophie
VL1
Geschichte
Personen
KI und Gesellschaft
Kritische Aspekte
Mathematik
VL2
Lineare Algebra
Statistik
Logik
Numerik
Analysis


KI-Landkarte
Künstliche Intelligenz
Anwendungen
Robotik
VL14
Computer Vision
VL9
Natürliche Sprache
VL10
Modellierung und Schlussfolgerung
Variablen
VL12
Inferenz
Logik
Wissensrepräsentation
VL11
Zustände
Suche
VL13
MDPs
Reflex
Lernen
Optimierung und Generalisierung
VL6
Vorhersage
VL4
VL5
Neuronale Netze
VL7
Modellierung
VL3
Supervised
Unsupervised
VL8
Historie und Philosophie
VL1
Geschichte
Personen
KI und Gesellschaft
Kritische Aspekte
Mathematik
VL2
Lineare Algebra
Statistik
Logik
Numerik
Analysis

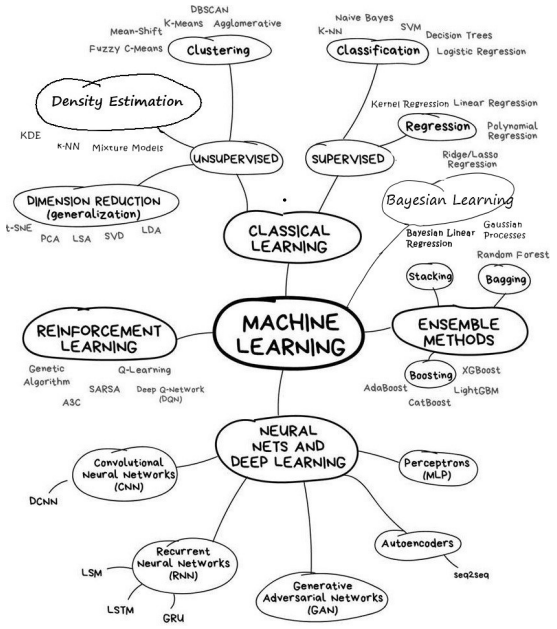
Überblick

1) Lernende Algorithmen

2) Bewertung von lernenden Algorithmen

3) Klassifikation, logistische Regression

Teil 1



1) Lernende Algorithmen

2) Bewertung von lernenden Algorithmen

3) Klassifikation, logistische Regression

Lernende Algorithmen - Definition

- Wie könnte man „Lernen“ definieren?
- Welche Begriffe müssen dazu wir definieren?

Lernende Algorithmen - Definition

Mitchell 1997:

> “A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*.“

> “Ein Computerprogramm kann als lernend bezeichnet werden, wenn es von Erfahrungen *E* bezüglich einer Klasse von Aufgaben aus *T* mit Leistungsmaß *P* lernt, und die Leistung in Aufgaben aus *T*, gemessen durch *P*, mit Erfahrung *E* zunimmt.

Was sind Beispiele für ***T* (tasks)**, ***P* (performance)** und ***E* (experience)**?

Aufgaben (tasks), *T*

- Klassifikation
  - *k* Kategorien von Input Vektoren: $f : \mathbb{R}^n \to \{1, \ldots, k\}$
  - Beispiele: Objekterkennung in Bildern
- Regression
  - Vorhersage von Werten: $f : \mathbb{R}^n \to \mathbb{R}$
  - Beispiele: Vorhersage von Mietpreisen
- Transkription
  - Unstrukturierter Input zu Text
  - Beispiele: Handschrift → Text, Gesprochene Sprache → Text
- Übersetzung
  - Symbole einer Sprache → Symbole einer anderen Sprache
  - Beispiel: Natürliche Sprachen
- Ausreißer Erkennung
  - Kreditkartentransaktionen
- Synthese
  - Textgenerierung

Leistungsmaß (performance), *P*

- Benötigt zur **quantitativen Analyse der Leistung** eines ML Algorithmus

- **Beispiele**
  - Klassifikation: Genauigkeit: Anteil der richtigen Vorhersagen
  - Regression: Maß für die Fehler der Vorhersagen

- **Allgemein**: Fehler auf **ungesehenen Daten**

  → Testset notwendig

  Die Daten im Testset sollen aus der selben Verteilung kommen, wie die Daten auf die das Modell später angewendet werden soll

- Die negative Performance (Fehlermaß) wird oft als „Loss“ bezeichnet

Erfahrung, *E*

- Typischerweise: **Datensatz** („dataset“)
  - Besteht aus *m* Datenpunkten, Samples
  - Oft: Jeder Datenpunkt ist ein Vektor mit *n* Einträgen
  - Jeder Eintrag wird als Attribut oder Feature bezeichnet
  - Design Matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$: Ein Sample pro Zeile

- Je nach Typ von Daten: Einteilung der Lernverfahren in überwachte („supervised“) und unüberwachte („unsupervised“) Tasks

  - **Supervised**: Jeder Datenpunkt hat ein Label oder einen Zielwert („target“)

$$p(\mathbf{y}|\mathbf{x})$$

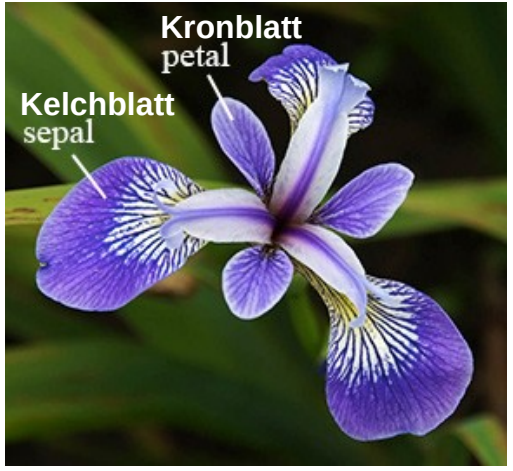
  - **Unsupervised**: Keine Labels oder Zielwerte

$$p(\mathbf{x})$$

Beispiel (überwacht): Iris-Datensatz

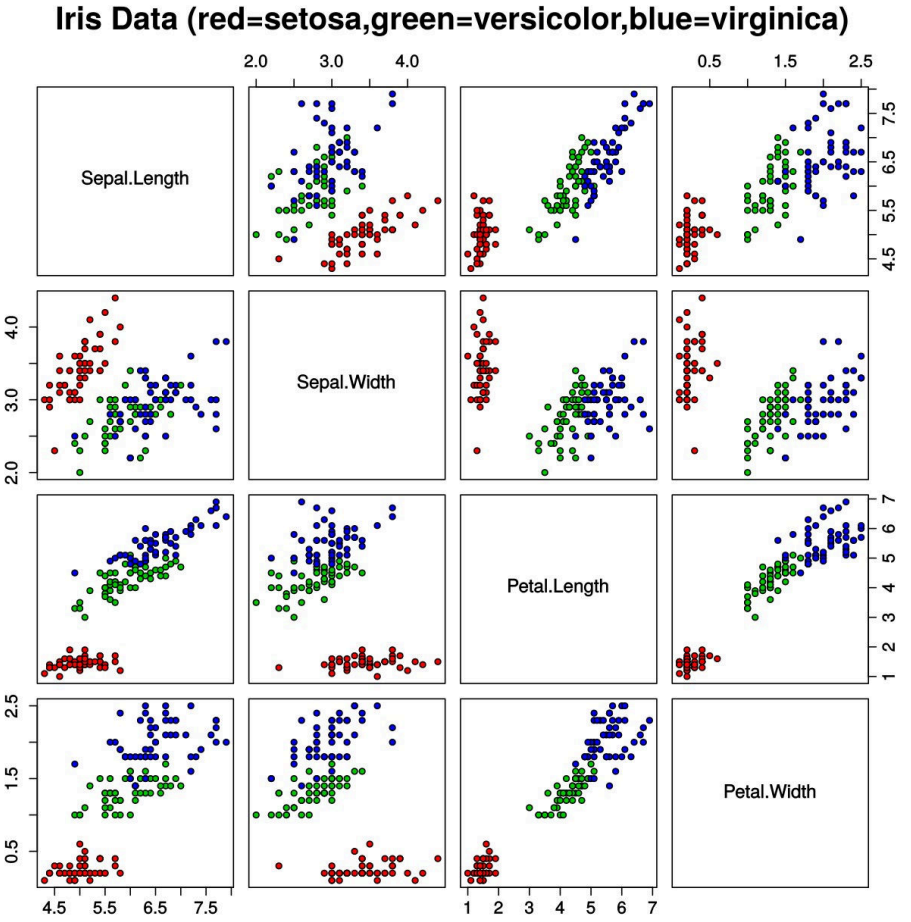
Einer der ältesten Datensätze
150 Datenpunkte/Samples
3 Klassen, 4 Attribute/Features

$$p(\mathbf{y}|\mathbf{x})$$

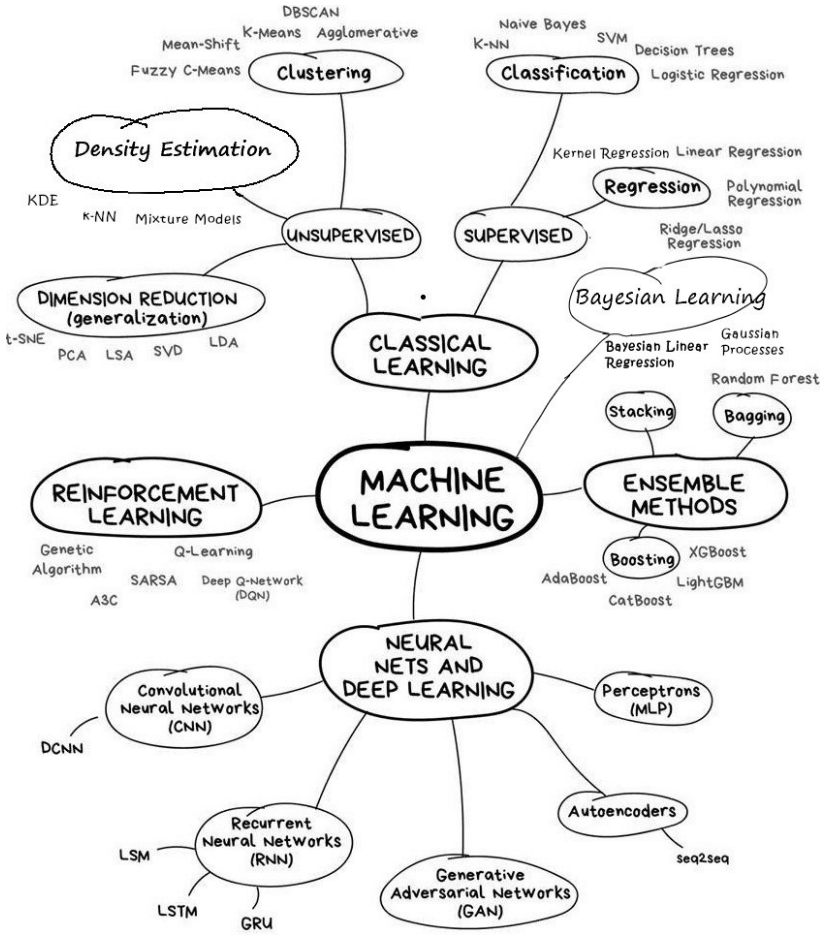



```
from sklearn.datasets import load_iris

iris = load_iris()
```



Machine Learning: Viele Begriffe und Methoden



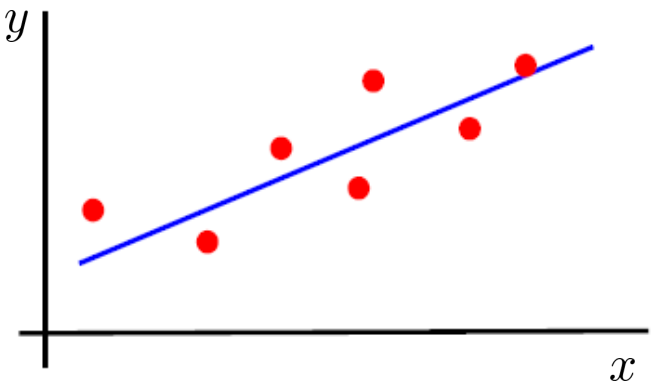
Gliederung nach **Lernmethoden** und Datentypen

- **Lernmethoden und Datentypen**
  - **Supervised Learning** *Daten mit Labels*
    - Regression
    - Klassifikation
  - **Unsupervised Learning** *Daten ohne Labels*
    - Clustering
    - Dimensionsreduktion
    - Density Estimation
  - **Reinforcement Learning** *Daten aus Interaktion*

Supervised Learning

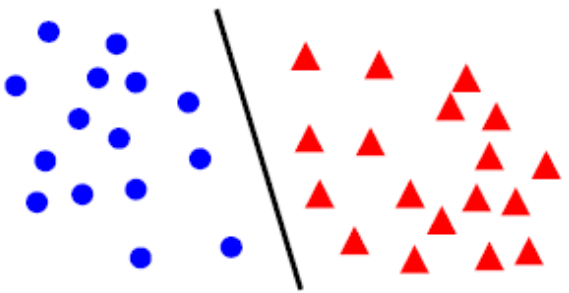
1. Regression

- Daten: Input x und Output („Labels“) y
- Lernen einer kontinuierlichen Funktion $y = f(x)$
- Beispiele: Lineare Regression



2. Klassifikation

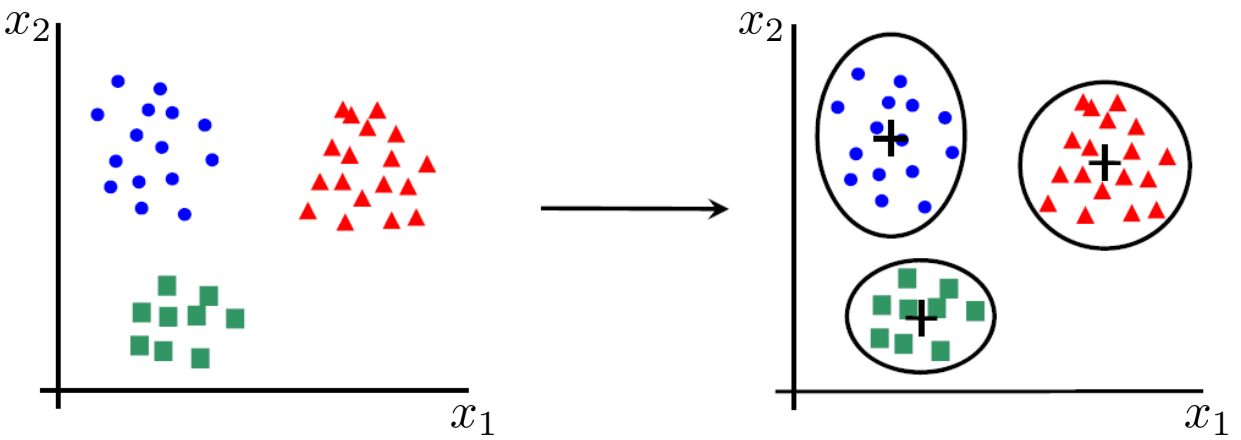
- Daten: Input x und Outputklassen („Labels“) c
- Lernen der Klassenzuordnung $c = f(x)$
- Beispiele: Handgeschriebene Zahlen



Unsupervised Learning

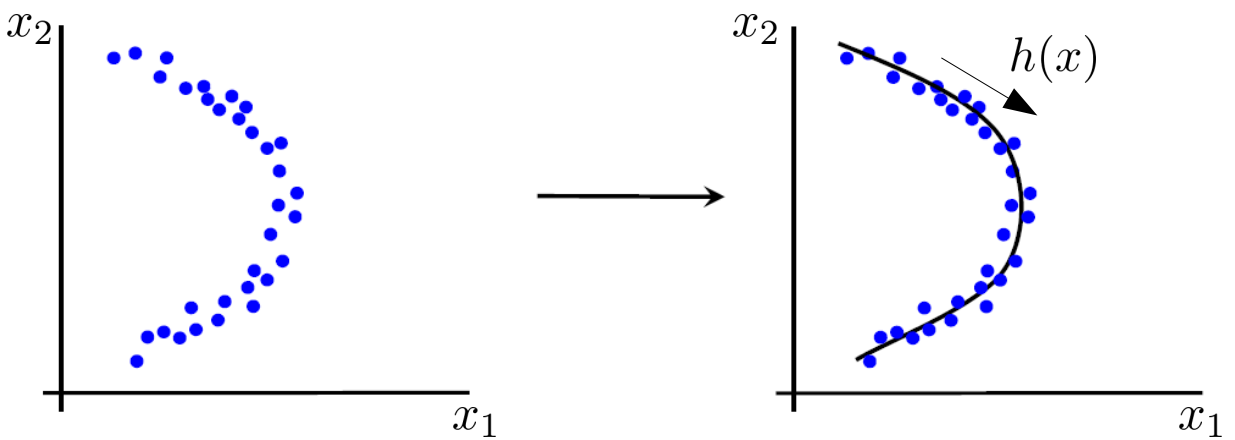
1. Clustering

- Daten: x
- Lerne mögliche Klassen c



2. Dimensionalitätsreduktion

- Daten: hochdimensionale x
- Lerne niedrigdimensionale h
- Lerne Funktion $h(x)$ und $x'(h)$



Supervised vs. Unsupervised Learning

- Keine klare Unterscheidung
- Unüberwachtes Problem kann in *n* überwachte Probleme umformuliert werden

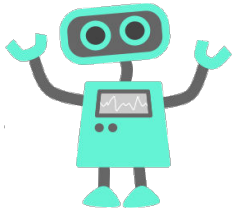
$$p(\mathbf{x}) = p\left(\mathbf{x}_0\right) p\left(\mathbf{x}_1|\mathbf{x}_0\right) p\left(\mathbf{x}_2|\mathbf{x}_0, \mathbf{x}_1\right) \ldots = \prod_{i=1}^{n} p\left(\mathbf{x}_i|\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}\right)$$

- Überwachtes Problem $p(y|\mathbf{x})$ kann mit unüberwachtem Problem $p(\mathbf{x}, y)$ gelöst werden

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}, y)}{\sum_{y'} p\left(\mathbf{x}, y'\right)}$$

- Weitere Algorithmen können gar nicht eindeutig eingruppiert werden:
  - Bestärkendes Lernen (Reinforcement Learning) → Nächste Folie

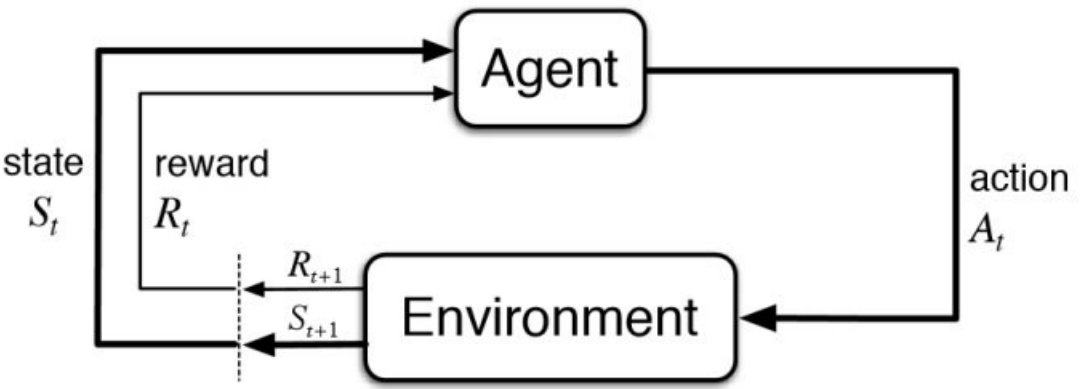
Reinforcement Learning

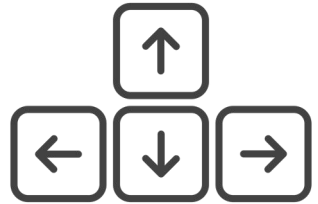


Belohnung am Ende

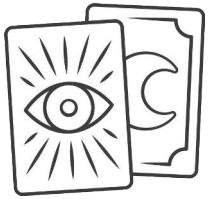
Unvollständig

Belohnung zwischendurch

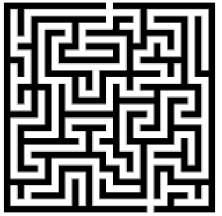




Diskret oder kontinuierlich



Orakel



Vollständiger Zustand

Eventuell stochastisch

Gliederung nach Modelltypen

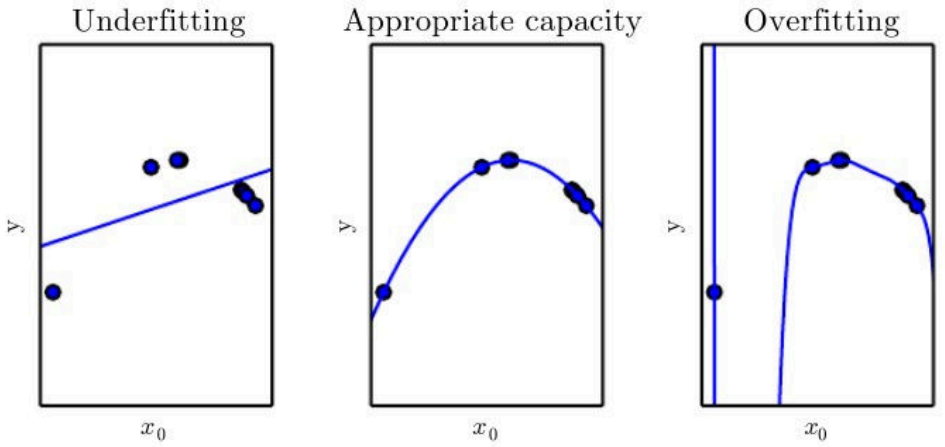
- **Modelltypen**
  - **Symbolische Modelle**
    - Entscheidungsbäume
  - **Neuronale Netze, Deep Learning**
    - Multilayer Perceptrons
    - CNNs
    - RNNs
    - Autoencoder
    - GANs
  - **Kernel-Methoden**
    - Support Vektor Maschinen
    - Gaussian Processes

Gliederung nach Modelltypen

- **Modelltypen**
  - **Symbolische Modelle**
    - Entscheidungsbäume
  - **Neuronale Netze, Deep Learning**
    - Multilayer Perceptrons
    - CNNs
    - RNNs
    - Autoencoder
    - GANs
  - **Kernel-Methoden**
    - Support Vektor Maschinen
    - Gaussian Processes

In dieser Vorlesung
Heute: Lineare Modelle

Teil 2



1) Lernende Algorithmen

2) Bewertung von lernenden Algorithmen

3) Klassifikation, logistische Regression

Kurze Erinnerung: Lineare Regression als ML Modell

- Gegeben: *m* Samples $\boldsymbol{x} \in \mathbb{R}^n$ und *m* Zielwerte $y \in \mathbb{R}$
- Modell: $\hat{y} = \boldsymbol{w}^\top \boldsymbol{x}$ mit Parametervektor $\boldsymbol{w} \in \mathbb{R}^n$
- Task *T*: Sage y für jedes $\boldsymbol{x}$ voraus.
- Leistungsmaß *P*:
  - Lege m_{test} Samples zurück: Testset $\mathbf{X}^{(\text{test})}$ und $\boldsymbol{y}^{(\text{test})}$
  - Berechne Fehler auf dem Testset: $\text{MSE}_{\text{test}} = \frac{1}{m} \sum_i \left(\hat{\boldsymbol{y}}^{(\text{test})} - \boldsymbol{y}^{(\text{test})} \right)_i^2$

$$\text{MSE}_{\text{test}} = \frac{1}{m} \left\| \hat{\boldsymbol{y}}^{(\text{test})} - \boldsymbol{y}^{(\text{test})} \right\|_2^2$$

- Lösung durch Minimierung des Fehlers: $\nabla_{\boldsymbol{w}} \text{MSE}_{\text{train}} = 0$
- Lösung: Normalengleichung:

$$\boldsymbol{w} = \left(\boldsymbol{X}^{(\text{train})\top} \boldsymbol{X}^{(\text{train})} \right)^{-1} \boldsymbol{X}^{(\text{train})\top} \boldsymbol{y}^{(\text{train})}$$

Randnotiz: Bias in der linearen Regression

- Lineare Regression enthält im Allgemeinen auch einen Bias-Term: $\hat{y} = \boldsymbol{w}^\top \boldsymbol{x} + b$
- Kann auch mit Normalengleichung gelöst werden:

$$\boldsymbol{x} = (x_1, ..., x_n) \longrightarrow \boldsymbol{x} = (x_1, ..., x_n, 1)$$

$$\boldsymbol{\omega} = (\omega_1, ..., \omega_n) \longrightarrow \boldsymbol{\omega} = (\omega_1, ..., \omega_n, b)$$

Generalisierung

- Herausforderung des maschinellen Lernens: **Generalisierung**
- → Gute Voraussagen auch für ungesehene Daten (Fehler auf dem Testset)
  - **Generalisierungsfehler**
- Hauptunterschied zwischen ML und Optimierung
- Beispiel: Lineare Regression
  - Training: Minimiere $$\mathrm{MSE}_{\mathrm{train}} = \frac{1}{m^{(\mathrm{train})}} \left\| \boldsymbol{X}^{(\mathrm{train})} \boldsymbol{w} - \boldsymbol{y}^{(\mathrm{train})} \right\|_2^2$$
  - Ziel: Minimiere $$\mathrm{MSE}_{\mathrm{test}} = \frac{1}{m^{(\mathrm{test})}} \left\| \boldsymbol{X}^{(\mathrm{test})} \boldsymbol{w} - \boldsymbol{y}^{(\mathrm{test})} \right\|_2^2$$
- Lösung nur möglich, unter der **Annahme**, dass beide Datensätze aus der gleichen Verteilung gezogen werden

Kapazität, Überfitten und Unterfitten

- Herangehensweise
    1. Minimiere den Fehler auf den Trainingsdaten
    2. Minimiere die Lücke zwischen Trainings und Testdaten

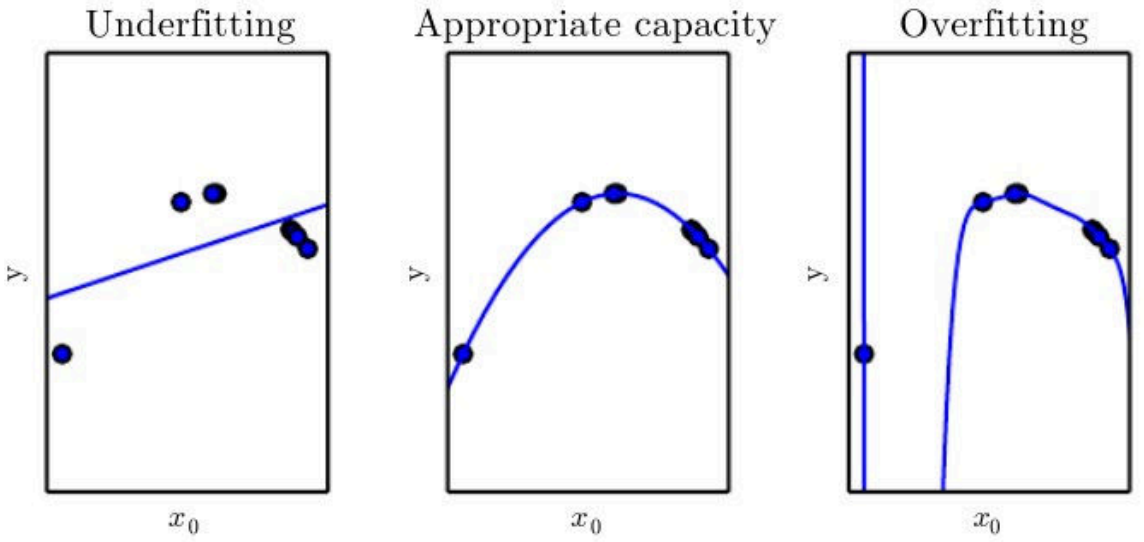
- Damit verbunden sind zwei grundlegende Konzepte des maschinellen Lernens:
    1. **Underfitting** → Großer Fehler auf Trainingsdaten
    2. **Overfitting** → Große Lücke zwischen Trainings- und Testfehler

- Kontrolle durch **Kapazität** eines Modells
    - Die Möglichkeit eines Modells, viele unterschiedliche Funktionen zu beschreiben
    - Niedrige Kapazität: Tendenz zum Unterfitten
    - Hohe Kapazität: Tendenz zum Überfitten

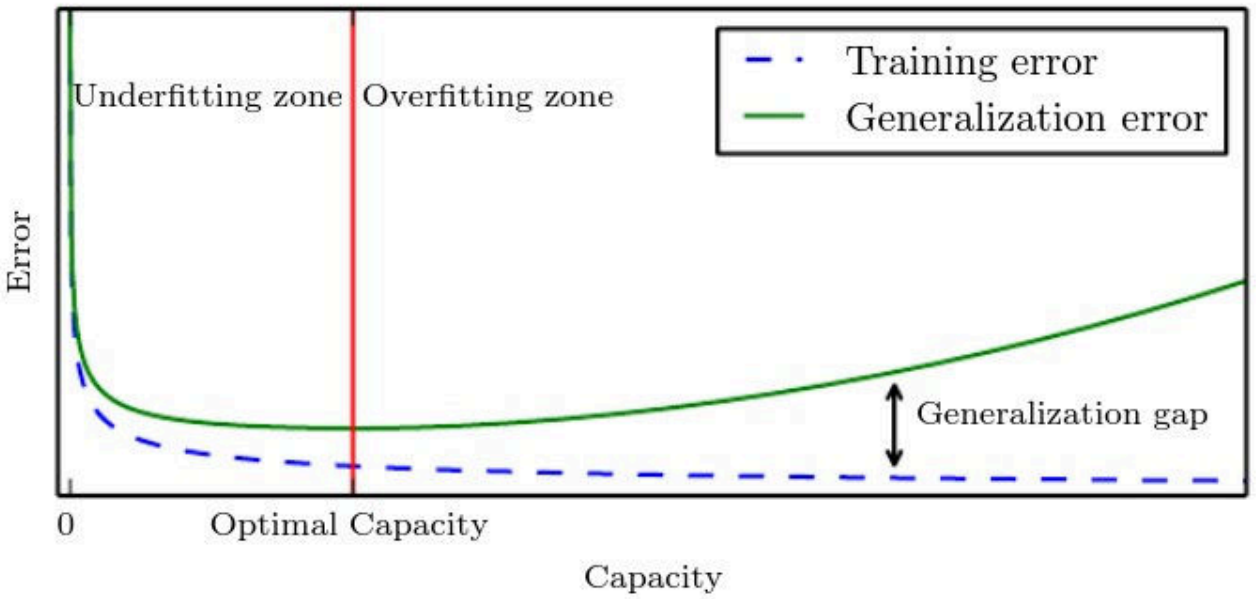
Kapazität, Überfitten und Unterfitten

- Beispiel

$$\hat{y} = b + wx \qquad \text{vs.} \qquad \hat{y} = b + w_1 x + w_2 x^2 \qquad \text{vs.} \qquad \hat{y} = b + \sum_{i=1}^{9} w_i x^i$$



Kapazität, Überfitten und Unterfitten



Begrenzung der Kapazität: Regularisierung

- Bisher: Kontrolle der Kapazität durch Änderung der zur Verfügung stehenden Funktionen
  - Benötigt Intuition und erhöht eventuell den Bias

- Alternativ: Modell muss sich selbst entscheiden
  - Bestrafung falls das Modell zu viele Funktionen benutzt
  - Quantisierung durch Gewichte („weight decay“)

$$\mathrm{Loss}(\boldsymbol{w}) = \mathrm{MSE}_{\mathrm{train}} \; + \lambda \boldsymbol{w}^\top \boldsymbol{w}$$

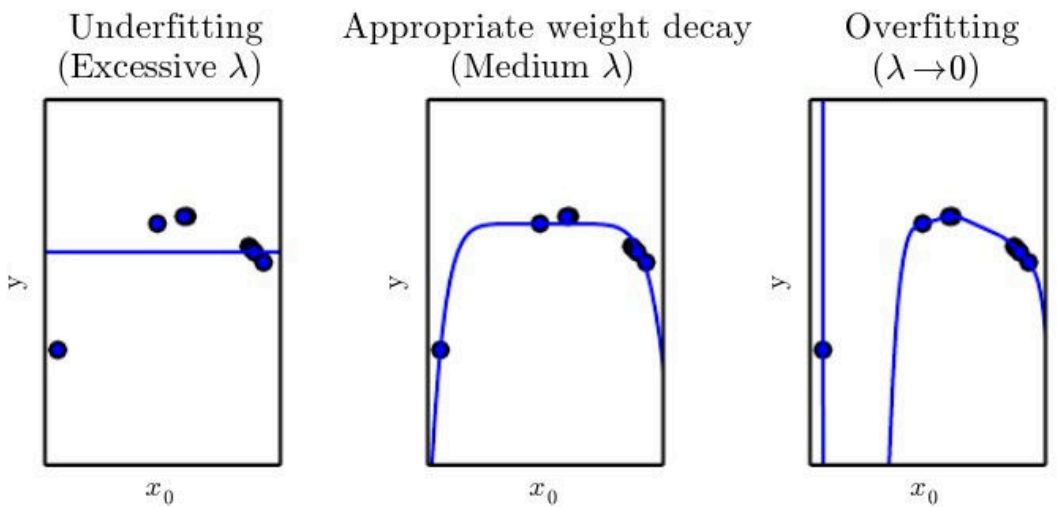
  - Präferiert
    - Kleinere Gewichte
      → Dadurch weniger steile Funktionen („Steilheit = Ableitung = Gewichte“)
      → Glattere/einfacherer Funktionen

Regularisierung

Beispiel

- Datenverteilung: Quadratische Funktion
- Fit durch Polynom 9ten Grades
- Variation des Regularisierungsparameters λ

$$J(\boldsymbol{w}) = \mathrm{MSE}_{\mathrm{train}} + \lambda \boldsymbol{w}^\top \boldsymbol{w}$$



Warum überhaupt MSE? → Maximum-Likelihood-Methode

- **Annahme:** Gute Schätzfunktion (Parameter $\boldsymbol{\theta}$) durch Minimierung von $\mathrm{MSE} = \frac{1}{m} \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_2^2$
- **Allgemeiner**
  - Methode der maximalen Plausibilität (Maximum-Likelihood-Methode)
- Gegeben
  - *m* Datenpunkte $\mathbb{X} = \left\{ \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)} \right\}$ verteilt mit unbekannter Verteilung $p_{\mathrm{data}}(\mathbf{x})$
- Gesucht
  - Parameter einer Wahrscheinlichkeitsfunktion $p_{\mathrm{model}}(\mathbf{x}; \boldsymbol{\theta})$

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} p_{\mathrm{model}}(\mathbb{X}; \boldsymbol{\theta})$$

$$= \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\mathrm{model}}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)$$

Maximum-Likelihood-Methode

- Lösung von: $\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\mathrm{model}}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)$
- Produkt durch Summe ersetzen → Logarithmus:

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\mathrm{model}}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)$$

- Teile durch *m*:

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(\boldsymbol{x}; \boldsymbol{\theta})$$

- Dies entspricht einer Maximierung der Ähnlichkeit von empirischer Datenverteilung $\hat{p}_{\mathrm{data}}(\boldsymbol{x})$ und $p_{\mathrm{model}}(\mathbf{x}; \boldsymbol{\theta})$

Maximum-Likelihood-Methode

- Für überwachtes Lernen: Suche $p_{\text{model}}\left(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}\right)$
- Damit: $\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} p_{\text{model}}\left(\boldsymbol{Y}|\boldsymbol{X};\boldsymbol{\theta}\right)$

  $$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}}\left(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right)$$

- Regression mit fehlerbehafteten Daten

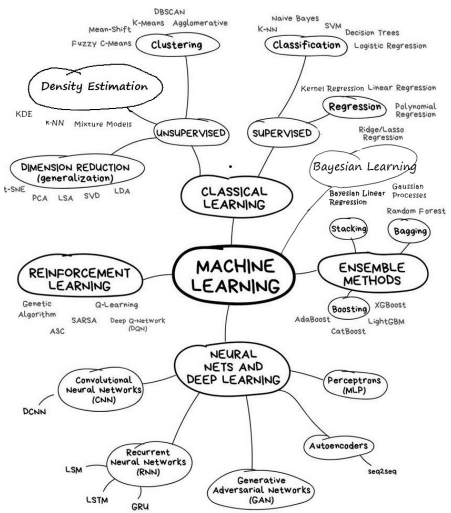
  $$p_{\text{model}}\left(y|\boldsymbol{x};\boldsymbol{\theta}\right) = \mathcal{N}\left(y;\hat{y}(\boldsymbol{x};\boldsymbol{\theta}),\sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{1}{2\sigma^2}\left(y-\hat{y}(\boldsymbol{x};\boldsymbol{\theta})\right)^2\right)$$

- Maximiere: $\sum_{i=1}^{m} \log p_{\text{model}}\left(y^{(i)}|\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right)$

  $$= -m\log\sigma - \frac{m}{2}\log(2\pi) - \sum_{i=1}^{m}\frac{\left\|y^{(i)}-\hat{y}(\boldsymbol{x}^{(i)};\boldsymbol{\theta})\right\|^2}{2\sigma^2}$$

- Äquivalent zu Minimierung von $\text{MSE}_{\text{train}} = \frac{1}{m}\sum_{i=1}^{m}\left\|y^{(i)}-\hat{y}(\boldsymbol{x}^{(i)};\boldsymbol{\theta})\right\|^2$

Zusammenfassung: Lernende Algorithmen und deren Bewertung

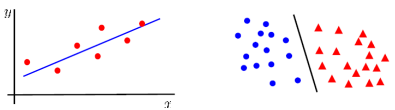


Tasks, Erfahrung, Performance

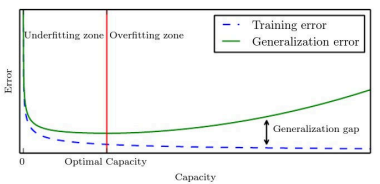
Überwacht vs. unüberwacht

$p(y|\mathbf{x})$ $p(\mathbf{x})$

Regression/Klassifikation



Generalisierung, Over/Underfitting



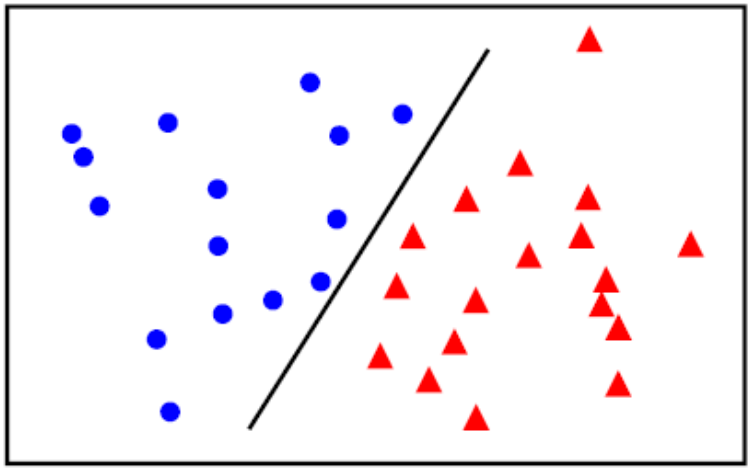
Regularisierung, Kapazität → Mehr in VL 9

$$J(\boldsymbol{w}) = \mathrm{MSE}_{\mathrm{train}} + \lambda \boldsymbol{w}^\top \boldsymbol{w}$$

Maximum Likelihood, MSE Loss

$$\mathrm{MSE}_{\mathrm{train}} = \frac{1}{m} \sum_{i=1}^{m} \left\| y^{(i)} - \hat{y}^{(i)} \right\|^2$$

Teil 3

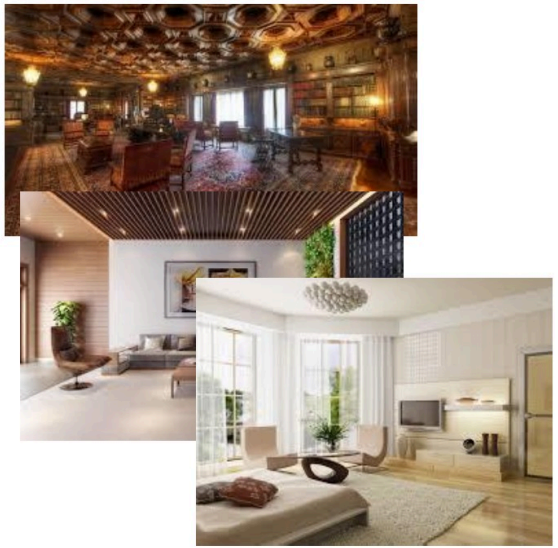


1) Lernende Algorithmen

2) Bewertung von lernenden Algorithmen

3) Klassifikation, logistische Regression

Beispiel: Klassifizierung von Bildern



Innen

Außen

(Bilddaten: Vorlesung 9)

Beispiel: Spam Erkennung

	#"$"	#"Mr."	#"sale"	...	Spam?
Email 1	2	1	1		Yes
Email 2	0	1	0		No
Email 3	1	1	1		Yes
...					
Email n	0	0	0		No
New email	0	0	1		??

Klassifikation: Formale Definition

Gegeben: Datensatz $\mathcal{D} = \{(\boldsymbol{x}_i, c_i)\}_{i=1\ldots N}$

mit Input Samples $\boldsymbol{x}_i \in \mathbb{R}^n$ und Klassen $c \in \{1 \ldots K\}$

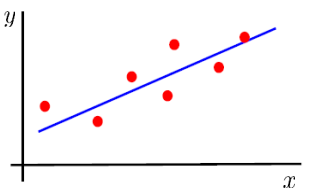
Ziel: Lernen einer Funktion $c = f(\boldsymbol{x})$

Nomenklatur: $K = 2$: Binäre Klassifikation

$K > 2$: Multiklassen Klassifikation

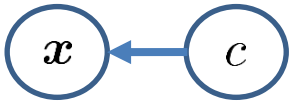
Vergleich: Regression: Kontinuierlicher Output

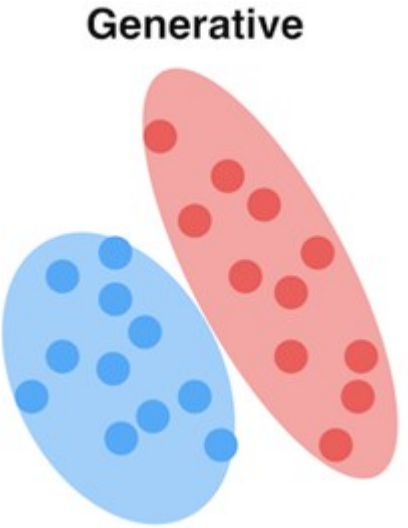
Klassifikation: Diskreter Output



Generative vs. diskriminative Modelle

Generative Modelle

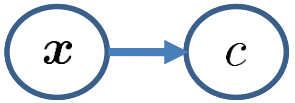




- Annahme: $p(c)$
- Lernen eines Modells $p(\boldsymbol{x} \mid c)$ aus den Daten
- Damit: Erzeugung von neuen Datenpunkten mit gegebenen Klassen
- Vorhersage der Klassen: $p(c \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid c)p(c)}{p(\boldsymbol{x})}$
- Das Lernen dieser Wahrscheinlichkeitsverteilungen ist sehr schwer!

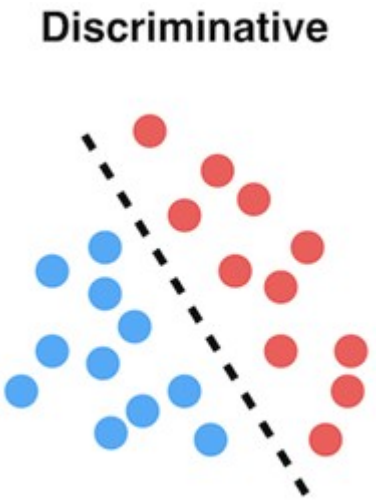
Generative vs. diskriminative Modelle

Diskriminative Modelle

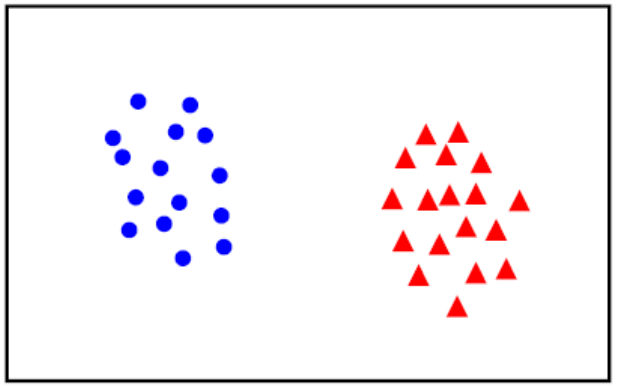


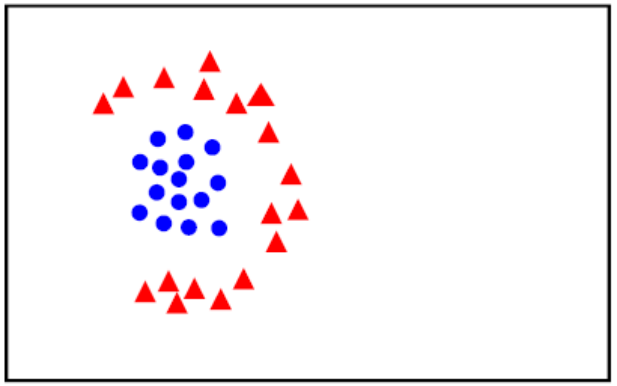
- Lernen einer Funktion $f(\boldsymbol{x})$ oder einer $p(c \mid \boldsymbol{x})$ Wahrscheinlichkeitsverteilung
- Die Funktion kann verschiedene $\boldsymbol{x}$ unterscheiden
- Modellierung muss die Punkte an der Grenze berücksichtigen
- Normalerweise sehr viel einfacher als Generation

  → Wir fokussieren uns heute auf diskriminative Modelle



Binäre Klassifikation





Diskriminative Modelle

- Daten: $(\boldsymbol{x}_i, c_i)\,, \mathrm{i} = 1 \ldots \mathrm{m}$ mit $\boldsymbol{x}_i \in \mathbb{R}^n$ und $c_i \in \{0, 1\}$
- Lerne einen Klassifikator:

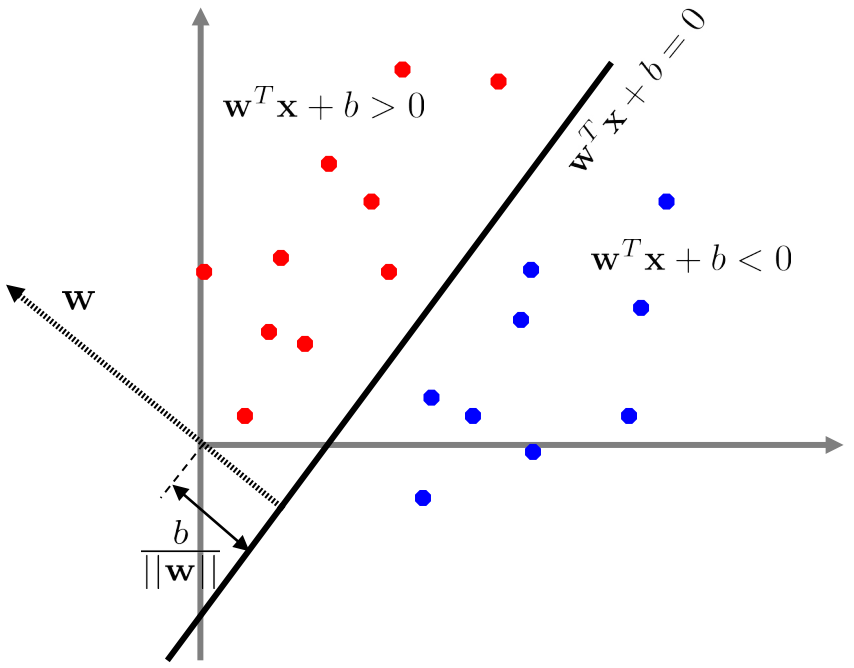
$$f\left(\boldsymbol{x}_i\right) = \begin{cases} > 0, & \text{if } c_i = 1 \\ < 0, & \text{if } c_i = 0 \end{cases}$$

Lineare Klassifikation

Linearer Klassifikator

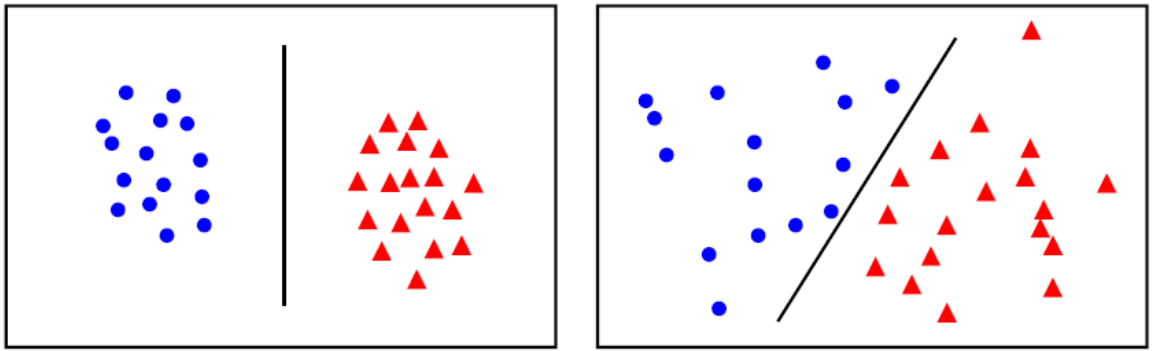
$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

- **In 2D: Linie**
  - $\mathbf{w}$ ist orthogonal zur Linie
  - b ist der y-Achsen-Abschnitt („bias“)
- **In 3D: Ebene**
- **In nD: Hyperebene**

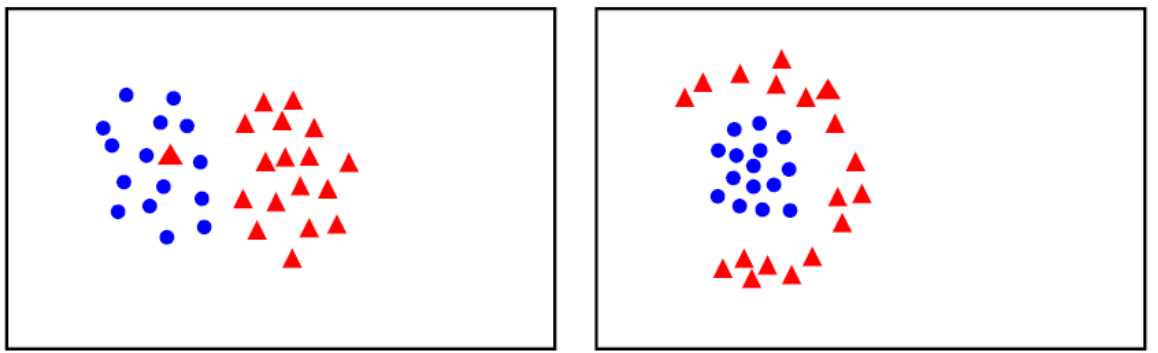


Lineare Separarabilität

Linear separierbar



Nicht linear separierbar



Lineare Klassifizierung: Erster Versuch

Vorhersage: $y = \text{step}(f(\boldsymbol{x})) = \text{step}\left(\boldsymbol{w}^T\boldsymbol{x} + b\right)$

- Falls $f(\boldsymbol{x}) < 0$: Klasse 0
- Falls $f(\boldsymbol{x}) \geq 0$: Klasse 1

Optimierung

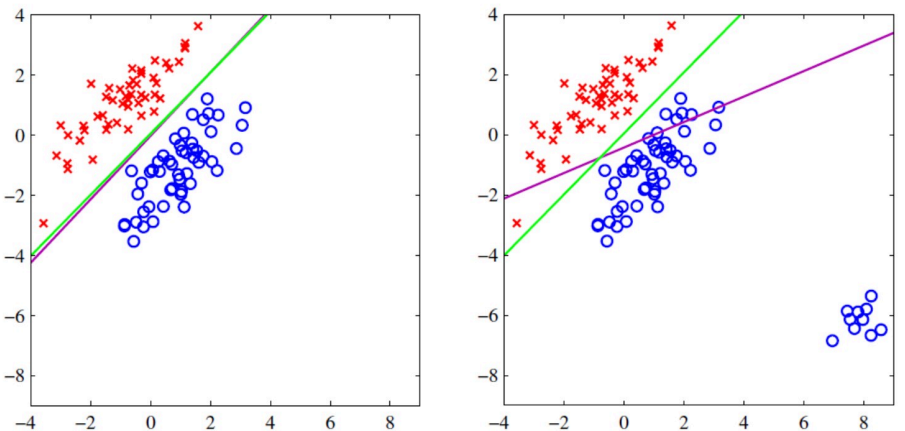
- Finde $\boldsymbol{w}$, sodass die Zahl der falsch klassifizierten Punkte minimal ist
- Quantifizierung: $L_0(\boldsymbol{w}) = \sum \mathbb{I}\left(\text{step}\left(\boldsymbol{w}^T\boldsymbol{x} + b\right) \neq c_i\right)$
- Sehr schwer zu optimieren! NP-schwer!

Lineare Klassifizierung: Zweiter Versuch

Können wir die gleiche Loss-Funktion wie für Regression benutzen?

$$L_{\text{reg}}(\boldsymbol{w}) = \sum_i \left(f\left(\boldsymbol{x}_i\right) - c_i\right)^2$$

- Minimierung der Loss-Funktion: Einfach!
- Aber: Die Labels sind nur 0 oder 1, der Output von $f\left(\boldsymbol{x}\right)$ ist kontinuierlich
- → Nicht robust gegenüber Ausreißern



Lineare Klassifizierung: Dritter Versuch

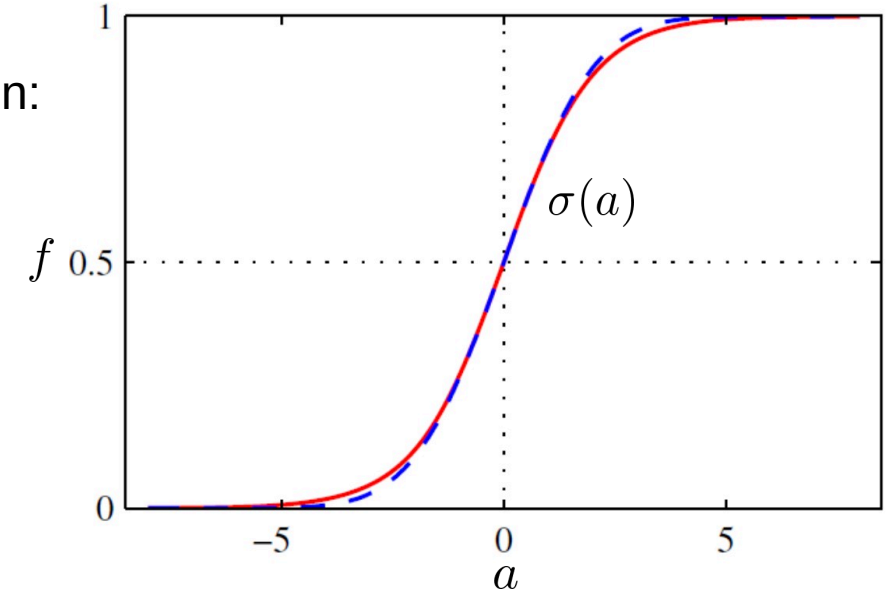
Können wir $f(x)$ einschränken?

- Benutze Sigmoid Funktion / logistische Funktion:

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

→ Output ist zwischen 0 und 1

→ Keine Step-Funktion, sondern glatt



Benutzung für Klassifikation

- Wende Sigmoid auf lineare Funktion an

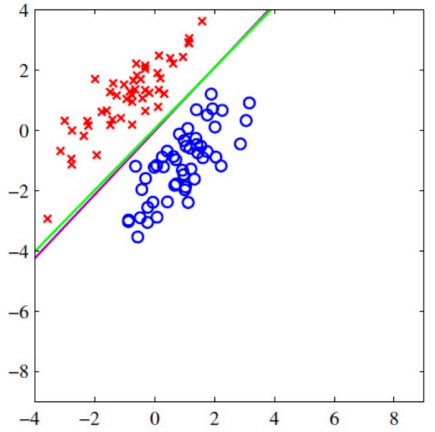
$$L(\boldsymbol{w}) = \sum_i \left(\sigma\left(f\left(\boldsymbol{x}_i\right)\right) - c_i\right)^2 = \sum_i \left(\sigma\left(\boldsymbol{w}^T \boldsymbol{x}_i + b\right) - c_i\right)^2$$

Besser: Probabilistische Betrachtung

Interpretation von $\sigma(f(x))$ als Wahrscheinlichkeitsfunktion:

$$p(c = 1 \mid \boldsymbol{x}) = \sigma\left(\boldsymbol{w}^T \boldsymbol{x} + b\right)$$

$$p(c = 0 \mid \boldsymbol{x}) = 1 - \sigma\left(\boldsymbol{w}^T \boldsymbol{x} + b\right)$$



Trick um dies in einer Formel zusammenzufassen: Label/Klasse *c* im Exponent

$$p(c \mid \boldsymbol{x}) = p(c = 1 \mid \boldsymbol{x})^c p(c = 0 \mid \boldsymbol{x})^{1-c} = \sigma\left(\boldsymbol{w}^T \boldsymbol{x} + b\right)^c \left(1 - \sigma\left(\boldsymbol{w}^T \boldsymbol{x} + b\right)\right)^{1-c}$$

(„Exponential-Trick“)

Log-Likelihood

Maximierung der Wahrscheinlichkeit → Maximierung des Logarithmus

$$\sum_i p\left(c_i \mid \boldsymbol{x}_i\right) \rightarrow \sum_i \log p\left(c_i \mid \boldsymbol{x}_i\right)$$

Wir können direkt die log-likelihood optimieren (maximieren)

$$\log \operatorname{lik}(\tilde{\boldsymbol{w}}, D) = \sum_i \log p\left(c_i \mid \boldsymbol{x}_i\right) = \sum_i \log \left(p\left(c=1 \mid \boldsymbol{x}_i\right)^{c_i} p\left(c=0 \mid \boldsymbol{x}_i\right)^{1-c_i} \right)$$

$$= \sum_i c_i \log p\left(c=1 \mid \boldsymbol{x}_i\right) + \left(1-c_i\right) \log p\left(c=0 \mid \boldsymbol{x}_i\right)$$

$$\tilde{\boldsymbol{x}}_i = [1, x_{i,0}, x_{i,1}, x_{i,2}, \ldots, x_{i,n}]$$

$$\tilde{\boldsymbol{w}}_i = [b, w_0, w_1, w_2, \ldots, w_n]$$

$$= \sum_i c_i \log \sigma\left(\tilde{\boldsymbol{w}}^T \tilde{\boldsymbol{x}}_i\right) + \left(1-c_i\right) \log \left(1-\sigma\left(\tilde{\boldsymbol{w}}^T \tilde{\boldsymbol{x}}_i\right)\right)$$

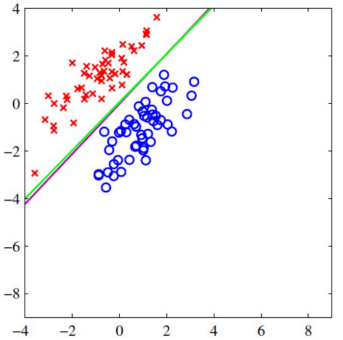
→ Negative log-likelihood wird oft als „**cross-entropy loss**“ bezeichnet

Logistische Regression

Optimierung der log-likelihood der Sigmoid-Funktion: „logistische Regression“

$$\text{argmax}_{\tilde{\boldsymbol{w}}} \log \text{lik}(\tilde{\boldsymbol{w}}, D) = \text{argmax}_{\tilde{\boldsymbol{w}}} \sum_i c_i \log \sigma\left(\tilde{\boldsymbol{w}}^T \tilde{\boldsymbol{x}}_i\right) + (1 - c_i) \log\left(1 - \sigma\left(\tilde{\boldsymbol{w}}^T \tilde{\boldsymbol{x}}_i\right)\right)$$

- … obwohl wir es für Klassifikation (nicht Regression) benutzen
- Die Loss-Funktion ist konvex → Nur ein Optimum
- Es gibt keine geschlossene Lösung (anders als bei linearer Regression)



Wie finden wir das Maximum? → **Gradient descent** (Gradientenabstiegsverfahren)

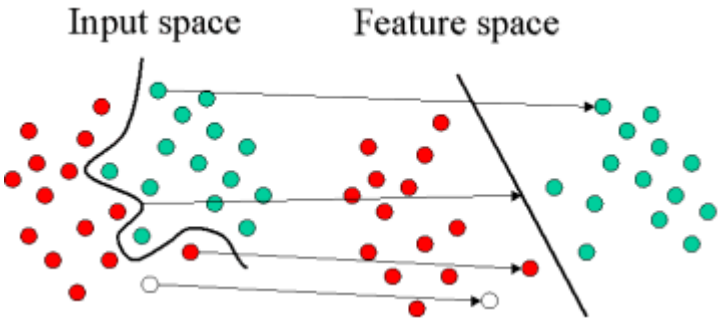
→ **Nächste Woche**

Verallgemeinerte logistische Regression

Linearer Klassifikator mit nicht-linearen Features $\boldsymbol{x}_i \rightarrow \phi\left(\boldsymbol{x}_i\right)$

$$\operatorname{argmax}_{\boldsymbol{w}} \log \operatorname{lik}(\boldsymbol{w}, D) = \operatorname{argmax}_{\boldsymbol{w}} \sum_i c_i \log \sigma\left(\boldsymbol{w}^T \boldsymbol{\phi}\left(\boldsymbol{x}_i\right)\right) + \left(1 - c_i\right) \log\left(1 - \sigma\left(\boldsymbol{w}^T \boldsymbol{\phi}\left(\boldsymbol{x}_i\right)\right)\right)$$

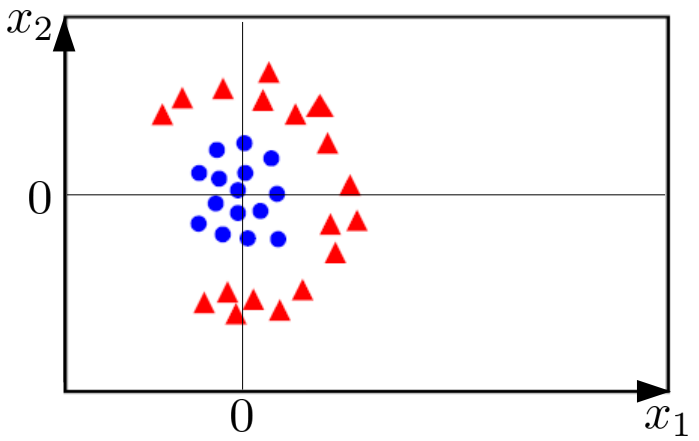
Ziel: Transformation der Punkte im Input-Raum (nicht linear separabel) zu Punkten im Feature-Raum die linear separabel sind



Verallgemeinerte logistische Regression

Ziel: Transformation der Punkte im Input-Raum (nicht linear separabel) zu Punkten im Feature-Raum die linear separabel sind

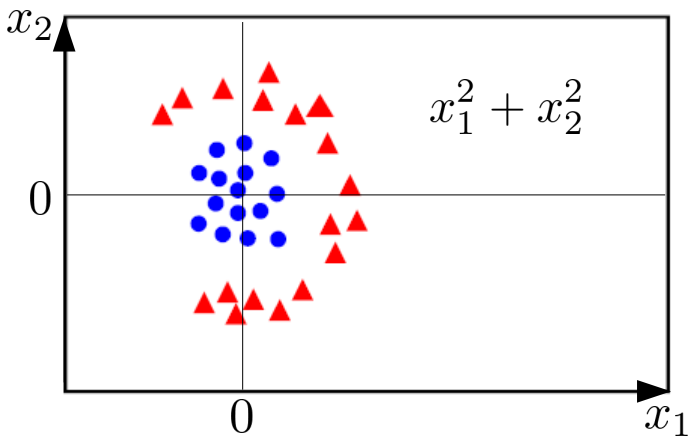
Beispiel



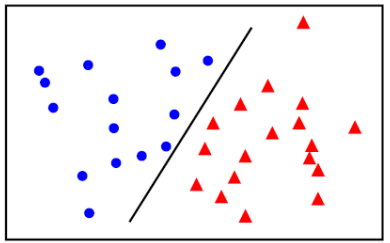
Verallgemeinerte logistische Regression

Ziel: Transformation der Punkte im Input-Raum (nicht linear separabel) zu Punkten im Feature-Raum die linear separabel sind

Beispiel



Zusammenfassung: Logistische Regression



Lineare Separierbarkeit

Sigmoidfunktion und logistische Regression

Log Likelihood Optimierung, Cross Entropy Loss

Zusammenfassung

1) Lernende Algorithmen

2) Bewertung von lernenden Algorithmen

3) Klassifikation, logistische Regression

→ Nächste Woche: **Nicht-lineare Probleme, neuronale Netze!**