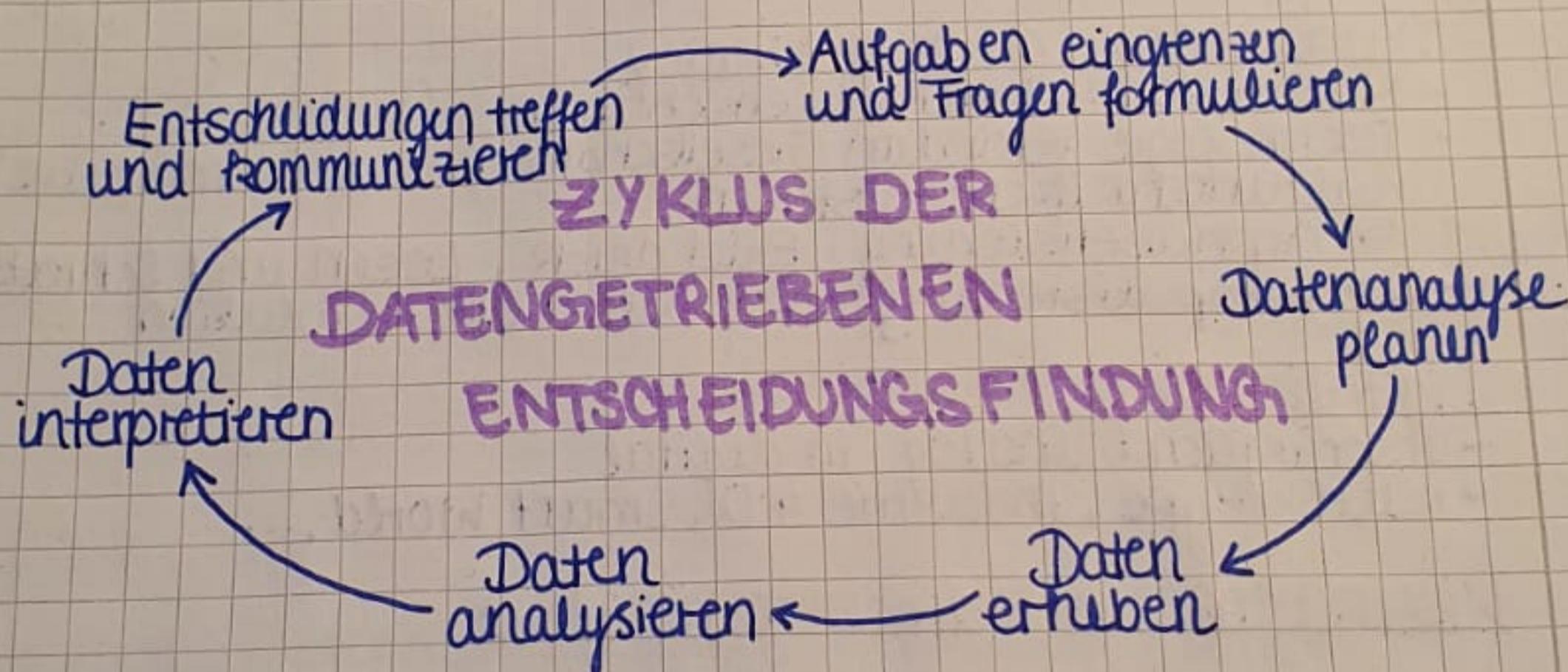
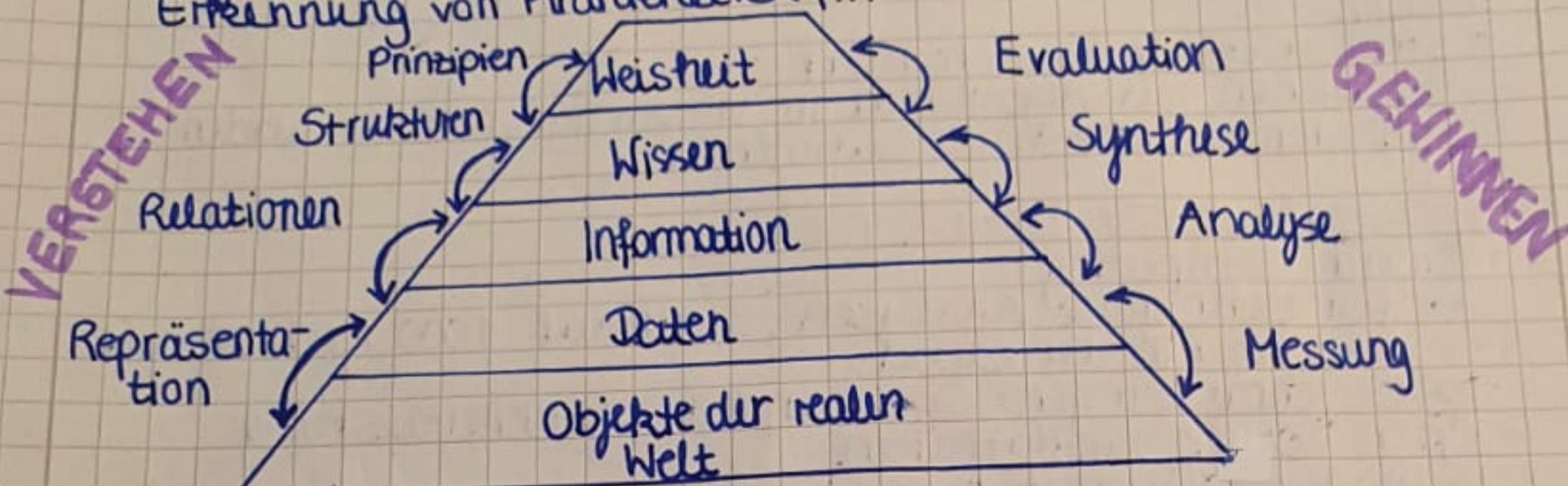


Maschinelles Lernen

Motivation für Maschinelles Lernen

- Daten sind heute überall vorhanden / ubiquitär
- Digitalisierung führt zu Datafizierung
 - ↳ Analoge in Digitale Welt
 - ↳ Daten als Nebenprodukt (durch Sensordaten, dig. Versicherungsdaten, Prozesse, Verkehrsflussdaten, ...)
- ermöglicht durch neue Analysemethoden neue Möglichkeiten
- Big Business für Assistenten in Smartphones, Spammer, Erkennung von Krankheiten, ...



Aus Daten lernen

- Tycho Brahe
 - ↳ ausgezeichneter Astronom, damals die genaueste Messung von Planetenbahnen ohne Fernrohr
 - ↳ hat Daten gesammelt aber keine Verwendung gehabt
- Johannes Keppler
 - ↳ Mathematikur
 - ↳ schlechter Beobachter, aber nahm Daten von Brahe
 - ↳ gab jedoch immer Abweichungen in seinen Vorhersagen, da das helozentrische Bild nicht gelangt hat
 - ↳ erstellte Ellipsen als Bahnlinien, mit der Sonne als Brennpunkt
 - ↳ Rudolphinische Tabellen: Vorhersage der Positionen der Planeten
 - ↳ Newton hat dadurch das Gravitationsgesetz erstellt
- was ist besser
 - ↳ leichterer Zugang an Daten
 - ↳ Berechnungen sind leichter zu implementieren
 - ↳ viel Rechenpower zum Testen von Algorithmen vorhanden

- was heute aber oft fehlt
 - ↳ (menschliche) Intuition
 - ↳ viele KI Methoden sind auf falschen Modellen

Was ist Machine Learning?

- Wissenschaft von Algorithmen, die den Sinn von Daten erkennen können
- Teildisziplin der künstlichen Intelligenz ohne menschliches Eingreifen, alles soll die Mathematik eifern
- Regeln werden nicht manuell abgeleitet
- Erfassung, welches Wissen in den Daten ist

Daten

- strukturiert und unstrukturiert
 - ↳ können mit Algorithmen in Wissen transformiert werden
 - ↳ Daten vorbereiten ist zeitintensiv

Infrastruktur für Data Science

- Cloud Computing
 - ↳ für sehr große Analysen, statt auf 1000 Rechner auf 1 Rechner in der Cloud
 - ↳ Beschränkungen: Speicherplatz, Rechenleistung
- Open Source Bibliotheken
 - ↳ in guter Qualität verfügbar

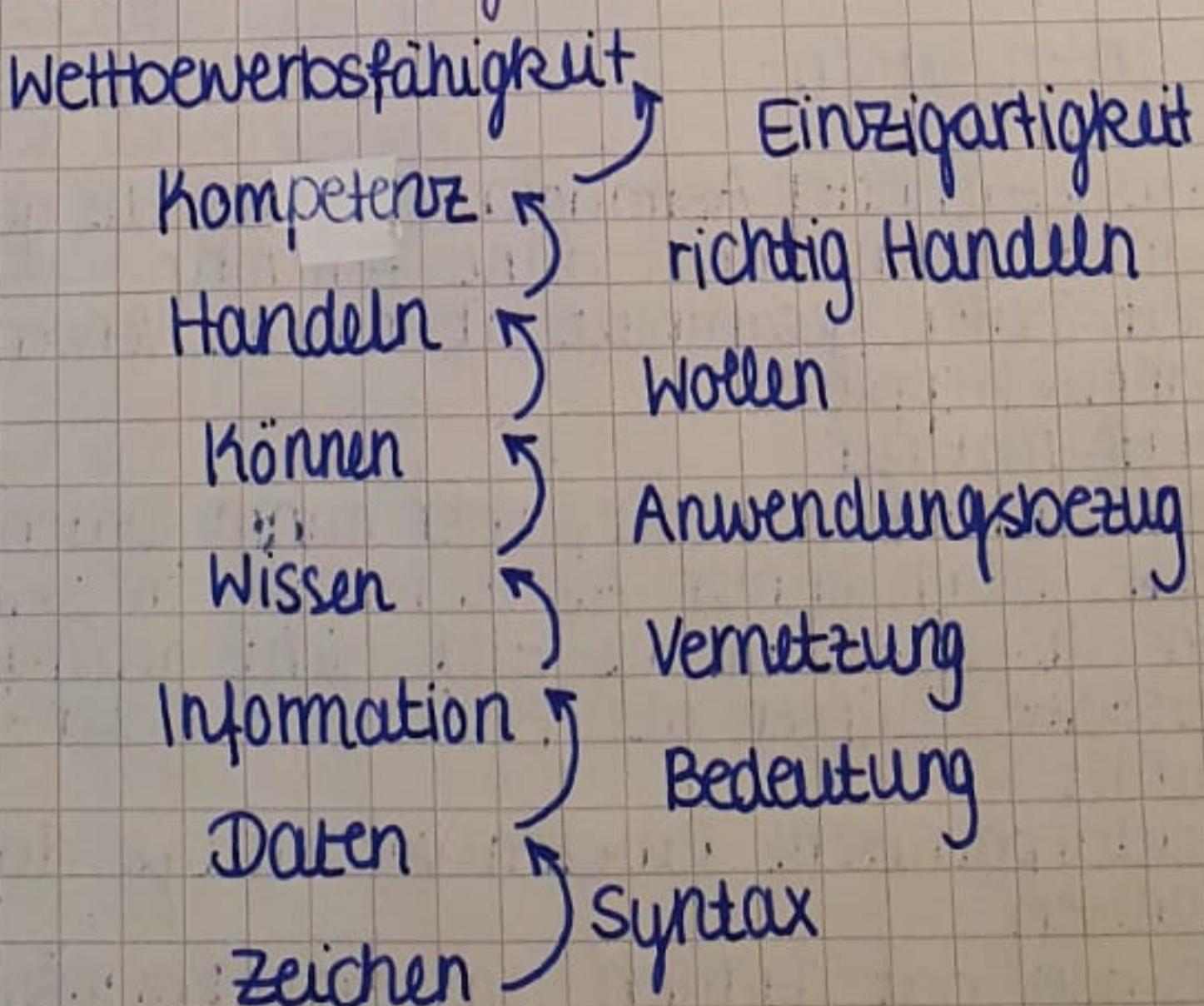
Data Science und Data Literacy

- Motivation: Datenwissenschaften
- Technologie verändert Gesellschaften und lassen neue Kulturtechniken entstehen
 - ↳ Alphabetisierung: Buchdruck, Lesen und Schreiben
 - ↳ Computisierung: 3. industrielle Revolution

Digitalisierung

- ursprünglich: analog in digital
- heute: Hype, Industrie 4.0, Smart World, ...

Wissenstreppe → Wettbewerbsfähigkeit



Nutzen von Data Science

- schnellere und bessere Entscheidungen treffen

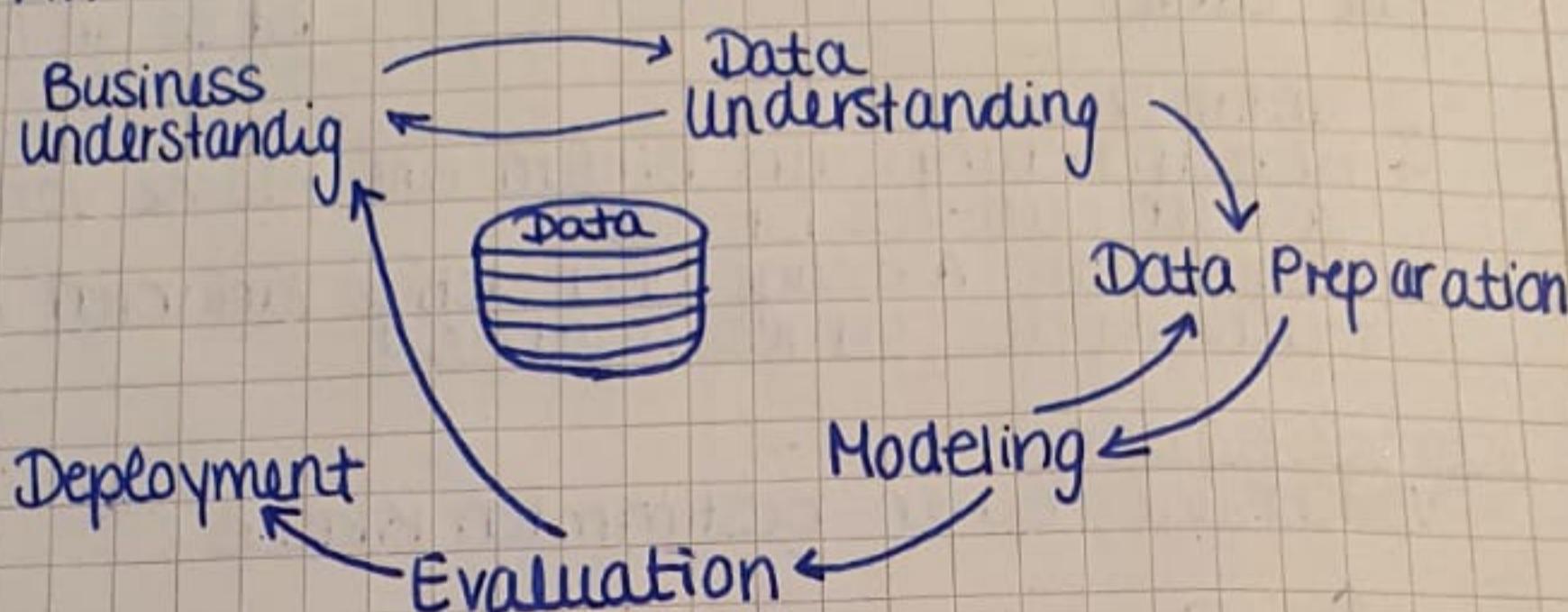
- Zeitgewinn und Wissensgrad nimmt zu

Datenzentrische Sicht

- bisher Prozesszentrische Sicht
 - ↳ Geschäftsprozesse mit Daten und Datenstrukturen
 - ↳ Daten sind Ergebnisse der Geschäftsprozesse
- jetzt Datenzentrische Sicht
 - ↳ Ergänzung der prozesszentrischen Sicht

Data Science als Prozess

- keine rein ingenieurtechnischer Zugang
- folgt einem explorativen, wissensgetriebenen, datenanalytischen Ansatz
- CRISP:



- ↳ Daten sind vielfältiger
- ↳ Analysen ändern sich, nicht nur eine Datenabfrage
- ↳ Mensch wird wichtiger

Veränderte Datenanalyse-Techniken

- Datenmengen nimmt zu
- Ressourcen-Auslastung ist schwer vorherzusagen
- Analyse gleich in Echtzeit

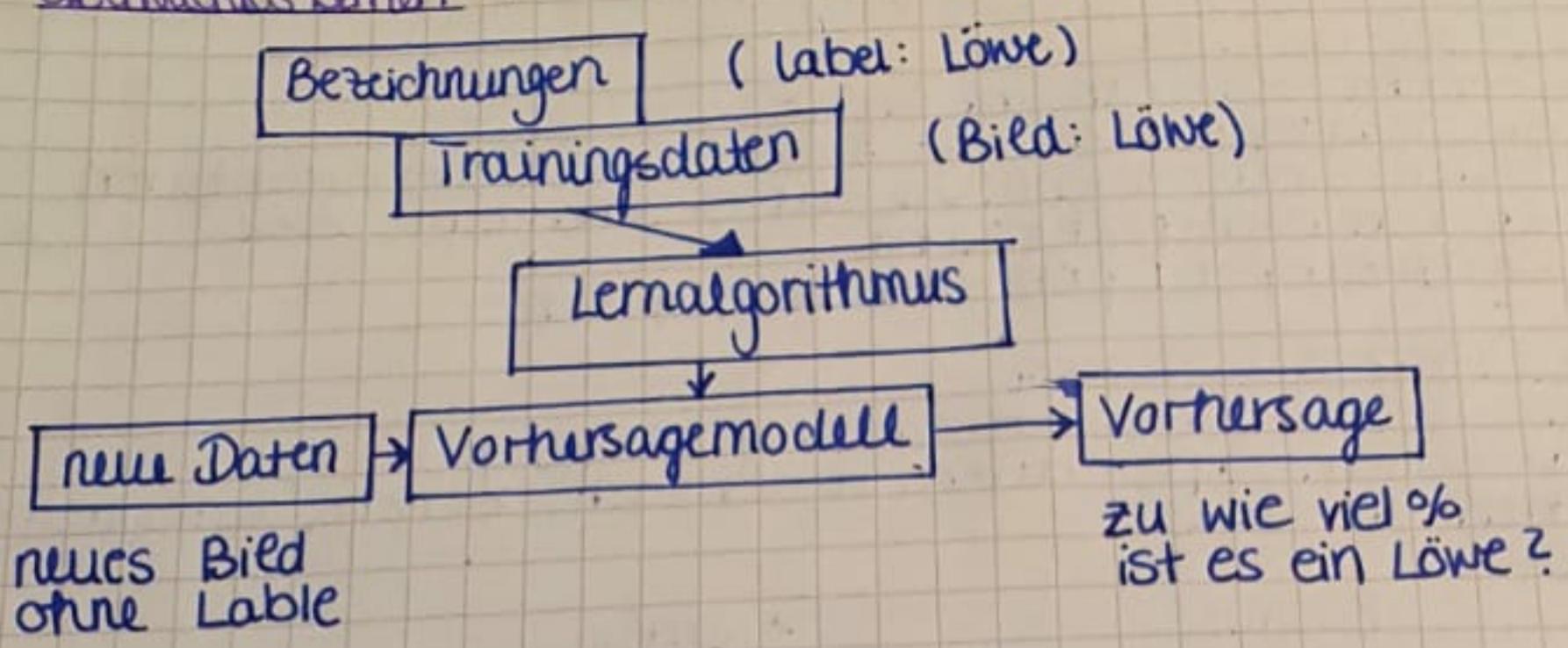
Big Data

- ~~B V~~
- Volumen: Datenmengen wachsen schneller als Datenzugriffsraten
- Velocity: große Datenmengen in kurzer Zeit verarbeiten
- Variety: keine feste Strukturen / keine Normalisierung und verschiedene Formatierungen

3 Arten von Machine Learning

- Überwachtes Lernen
 - ↳ Ziel: Vorhersage eines Ergebnisses / der Zukunft
 - ↳ Voraussetzung: gekennzeichnete Daten (z.B. Tierbilder)
 - ↳ Vorteile: direktes Feedback, ob das Lernen erfolgreich ist, gute Validierung möglich
- Unüberwachtes Lernen
 - ↳ keine Kennzeichnung der Daten
 - ↳ Ziel gibt es nicht, jedoch sollen verborgene Strukturen gefunden werden
 - ↳ kein Feedback möglich
- Verstärkendes Lernen
 - ↳ Ziel: Aktionen erlernen mit Belohnungssystemen
 - ↳ Rückführend auf Überwachtes Lernen

Überwachtes Lernen



- 2 Unterklassen
 - ↳ Klassifizierung: aus Bildern eine Klasse vorhersagen, ja oder nein-Jes ist das
 - ↳ Regression: Aussage über stetige Variablen (z.B. Berechnung des Jahresumsatzes)

Klassifizierung

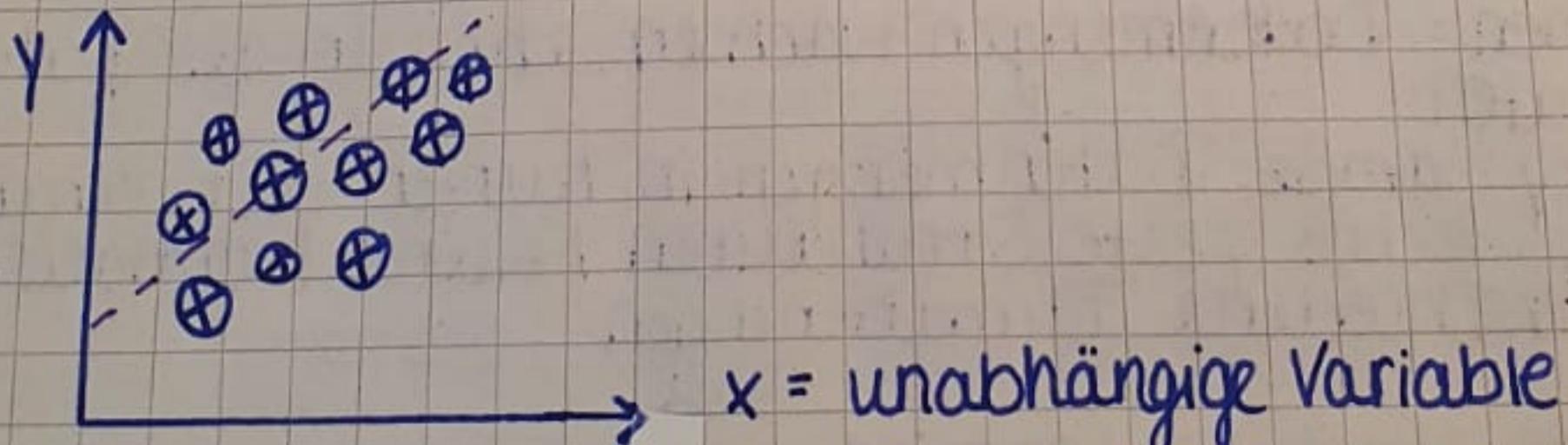
- Vorhersage einer bestimmten Klasse



- + Spam - kein Spam
- Abhängigkeit der Klassen
- Trennungslinie bestimmen

Regressionsanalyse

- Vorhersage stetiger Variablen
- unabhängige Variablen = Regressorvariablen
- eine oder mehrere Zielvariablen
- z.B. Vorhersage von Klausurergebnissen aus Übungsergebnissen



Verstärkendes Lernen

- Verwandtschaft mit überwachtem Lernen wegen des Belohnungszentrums
- Belohnungssignal ist ein Maß, wie gut die Aktion war (z.B. Züge beim Computer-Schach)

Unüberwachtes Lernen

- Suche unbekannter Struktur
- Informationen extrahieren ohne Zielvariable und ohne Belohnungsfunktion
- Methoden: Clustering, Datenkomprimierung

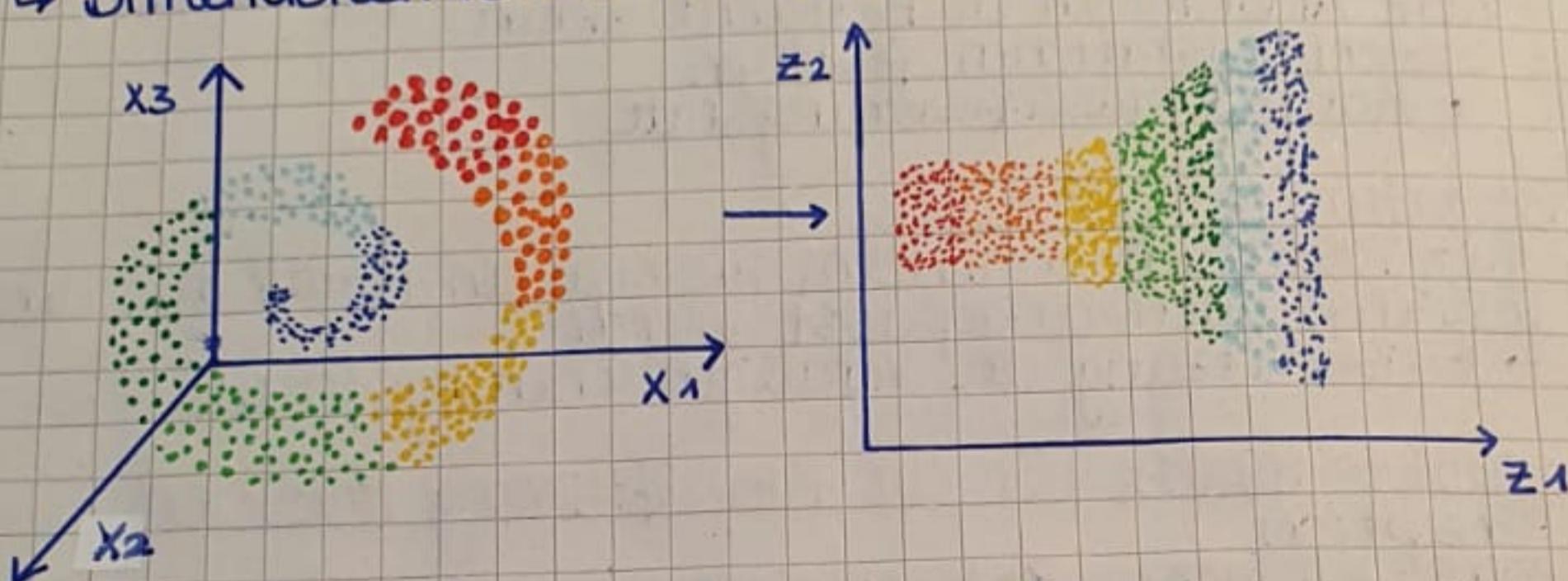
Clustering



- definieren Gruppen mit gemeinsamen Eigenschaften
- Gruppen unterscheiden sich voneinander
- unüberwachte Klassifizierung

Datenkomprimierung durch Dimensionsreduzierung

- hohe Datendimensionalität, es gibt viele Merkmale
- oft beschränkte Recheninfrastruktur
 - ↳ deswegen vorberechnen, um Speicher zu sparen
 - ↳ Dimensionen reduzieren durch Transformation



Mathematische Notation

- Algorithmus soll trainiert werden, dieser hat Parameter
- und damit soll eine Vorhersage gemacht werden

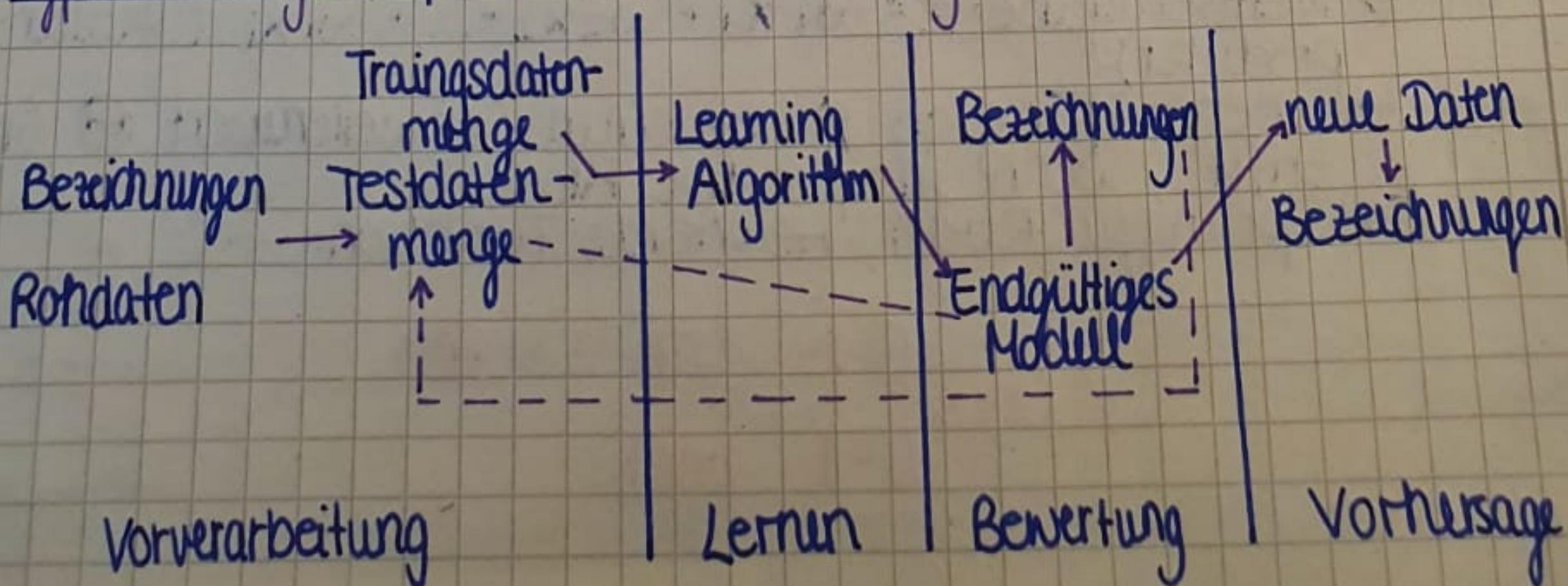
$$\vec{y} = F(\vec{x})$$

\vec{y} : Vektor vorhersagender Werte
 \vec{x} : Vektor mit Merkmalen eines Objekts

IRIS-Datensammlung

- sind Schwerlinien
- Datensammlung mit 150 Schwerlinien
- 3 Arten: Iris setosa, versicolor und virginica
- Datensatz wird in Merkmalmatrix X überführt
 - ↳ Spalten: Merkmale, Zeilen: Datensätze
- Zielvariable
 - ↳ besondere Bedeutung im Trainingsprozess
 - ↳ auf IRIS, nur die 3 Arten als Zielergebniss

Typisches System für Machine Learning



- Vorbereitung: Merkmalsextraktion und Maßstab, Merkmalauswahl, Dimensionsreduktion, Stichproben
- Lernen: Modellauswahl, Kreuzvalidierung, Leistungskriterien, Hyperparameter-Optimierung

Vorverarbeitung: Daten in Form bringen

- für IRIS:

1. Fotos der Blüten
2. normierte Merkmale
3. Korrelationen zwischen den Merkmalen
4. Dimensionsreduktion
 - ↳ Reduzierung des Rechens- und Speicherbedarfs
 - ↳ Rauschverhältnisse werden verbessert
5. Trennung in Trainings- und Testdaten

Trainieren und Auswählen eines Vorhersagemodells

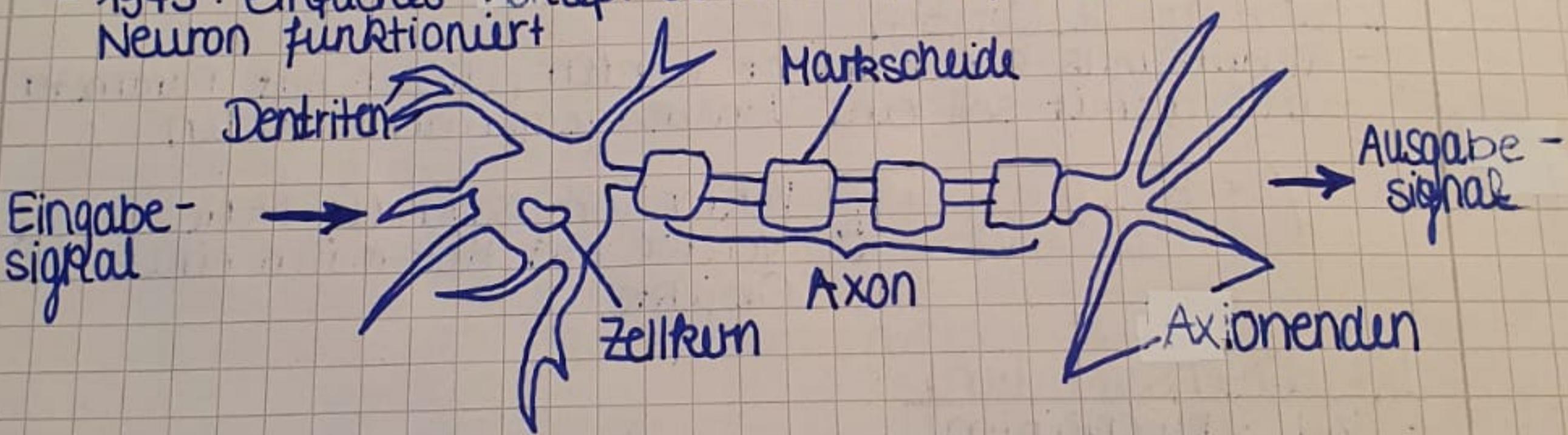
- viele Algorithmen in Betracht ziehen
- Bewertungskriterien festlegen
 - ↳ Korrekt Klassifizierungsrate

Bewertung von Modellen

- mit Testdaten ermitteln, wie gut ein Modell aus den Trainingsdaten angepasst wurde
 - ↳ Einschätzung des Verallgemeinerungsfehlers

Lernalgorithmen für die Klassifizierung trainieren

- Perzeptron
- 1943: einfaches Konzept einer Hirnzelle, wie das MCP Neuron funktioniert



- 1957: Rosenblatt erstellt Perzeptron - Lernregel
 - ↳ Ermittlung der Gewichtungskoeffizienten

Formelle Beschreibung eines künstlichen Neurons

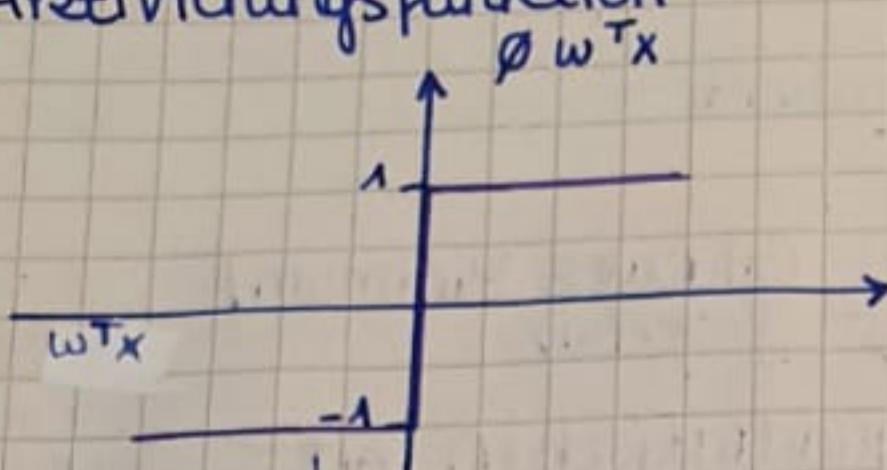
- binäre Klassifizierung aus 2 Klassen
 - ↳ positive Klasse +1
 - ↳ negative Klasse -1
- Gewichtungsvektoren w und Eingabeverketoren
- Aktivierungsfunktion berechnen $\phi(z)$
 - ↳ Nettoeingabe: $z = w_1 x_1 + w_2 x_2 + \dots + w_i x_i$ (Skalarprodukt)
 - ↳ Überprüfung ob $\phi(z)$ einen Schwellenwert Θ übertrifft, wenn ja = positive Klasse
 - ↳ Sprungfunktion / Heaviside Funktion:

$$\phi(z) = \begin{cases} 1 & \text{wenn } z \geq \Theta \\ -1 & \text{sonst} \end{cases}$$

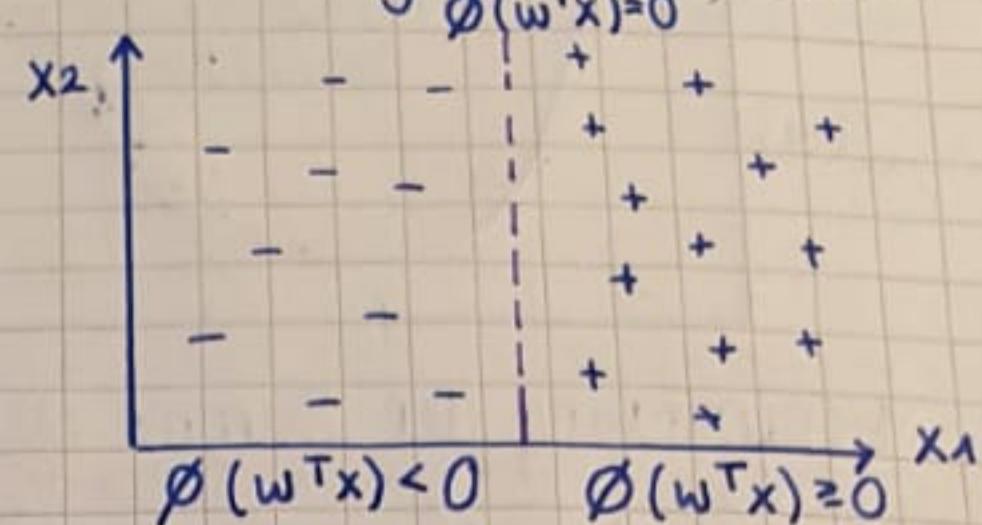
- Vereinfachung des Schwellenwertes Θ durch die Bias Einheit
 - $w_0 = -\Theta$
 - $x_0 = 1$

$$\phi(z) = \begin{cases} 1 & \text{wenn } z \geq 0 \\ -1 & \text{sonst} \end{cases}$$

- Aktivierungsfunktion:

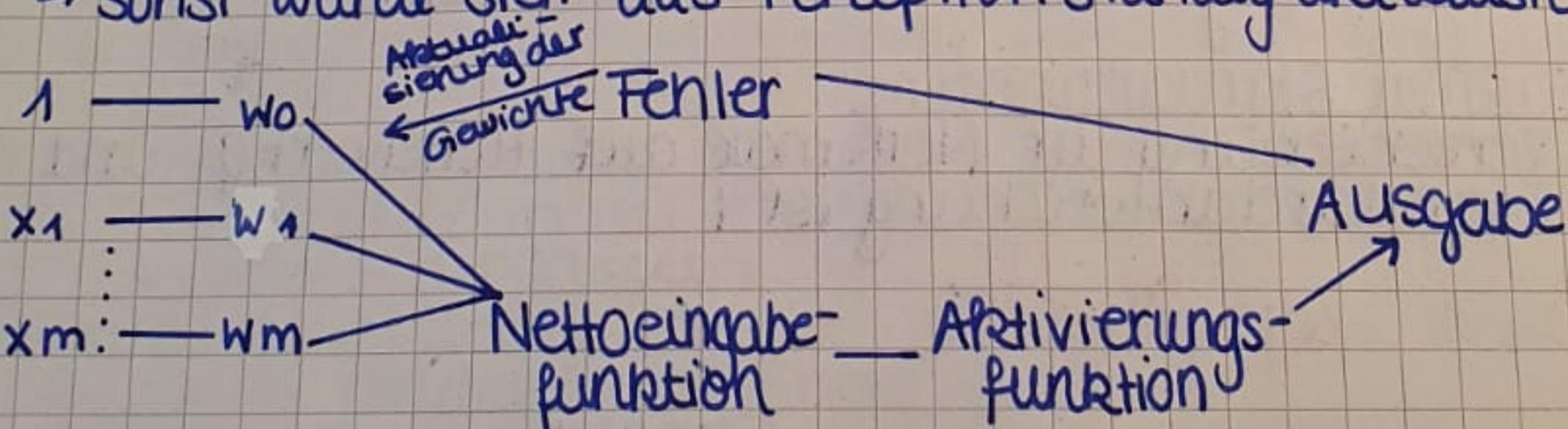


Trennung von 2 Klassen:



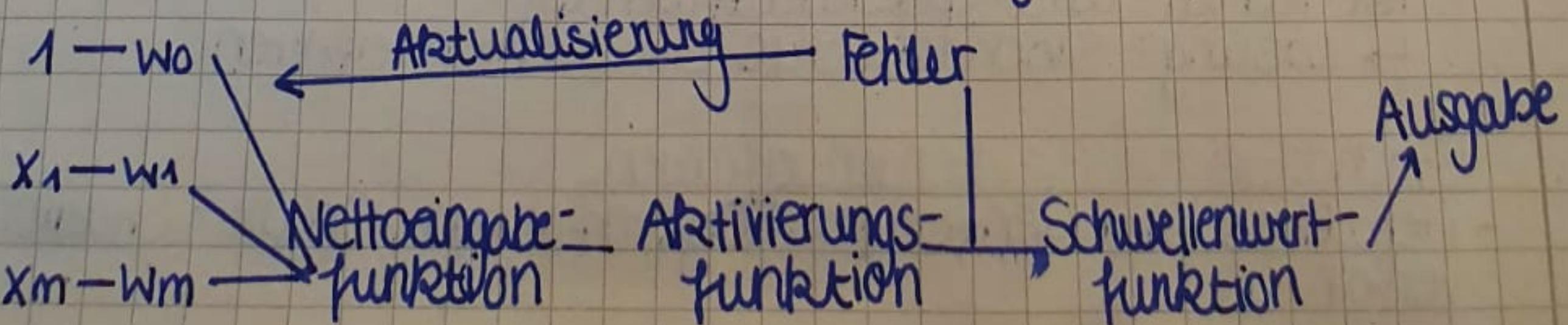
Lernregel für Perzeptrons

- Ziel: Ermittlung der Gewichte w anhand der bekannten Eingangsgrößen x und Ausgangsgrößen y
- Ansatz:
 - \hookrightarrow 1. Gewichte mit kleinen Zufallswerten = 0
 - \hookrightarrow 2. Trainingsdurchläufe mit Trainingsobjekten $x^{(i)}$ ausführen
 - \hookrightarrow Berechnung des Ausgabewertes \hat{y}
 - \hookrightarrow Aktualisierung der Gewichte $w_j := w_j + \Delta w_j$
 - $\Delta w_j = \eta (y^{(i)} - \hat{y}^{(i)}) x^{(i)}$
 - \downarrow Lernrate \uparrow Voraussage
- soll linear trennbar sein
- wenn es keine Entscheidungsgrenzen gibt, dann müssen eine max. Anzahl von Trainingsdurchläufen oder eine tolerierbare Fehlerklassifizierung festgelegt werden
 - \hookrightarrow sonst würde sich das Perzeptron ständig aktualisieren



Adaptive lineare Neuronen und Konvergenz des Lernens

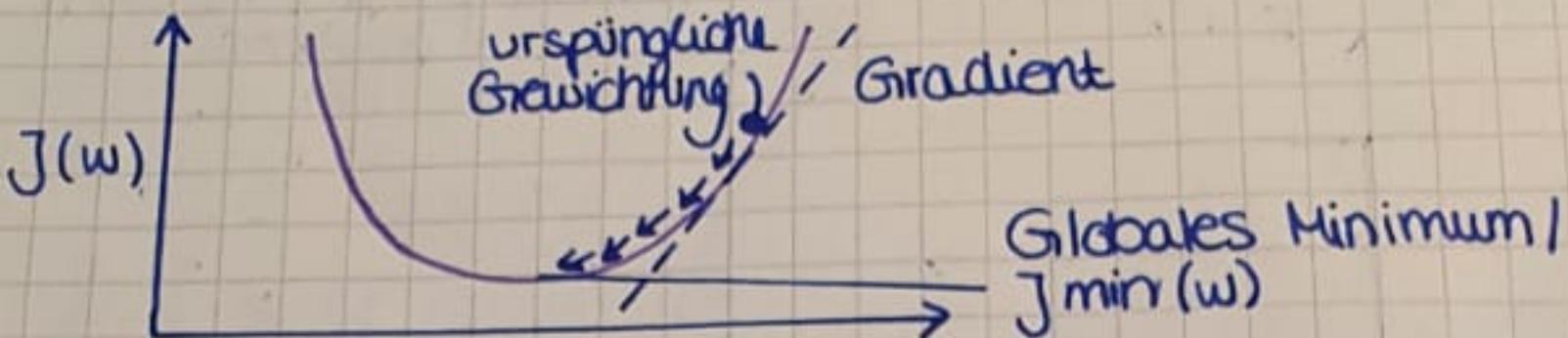
- weiteres neuronales Netz
- besteht nur aus einer Schicht
- Adaline verwendet nun eine lineare Aktivierungsfunktion $\phi(z) = \phi(w^T x) = w^T x$
 - \hookrightarrow um Gewichte zu erlernen, nicht mehr durch Sprungfunktion
 - \hookrightarrow Schwellenwertfunktion wird eingeführt



- Adaline Grundlagen: logische Regression, Support Vector Machine, Regressionsmodell

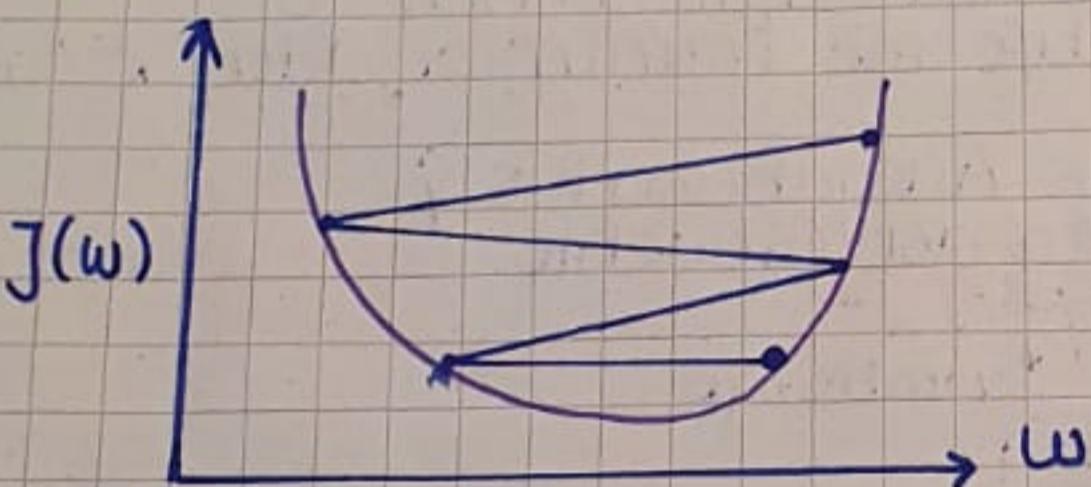
Optimale Anpassung eines Modells an vorhandene Daten

- Einführung einer Kostenfunktion $J(w)$
 - ↳ soll Abweichung zwischen Daten y und Vorhersagen $F(x)$ berechnen
 - ↳ optimal: Fehler klein halten
- Ansatz: Gradientenabstiegsverfahren



- Minimierung der Summe der quadratischen Abweichungen:

$$J(w) = \frac{1}{2} \sum (y^{(i)} - \phi(z^{(i)}))^2 = \frac{1}{2} \sum (y^{(i)} - \phi(w^T x^{(i)}))^2$$
 - ↳ Gradient bestimbar: $-\sum (y^{(i)}) - \phi(z^{(i)}) x_j^{(i)}$
 - ↳ Aktualisierung der Gewichte: $w := w + \Delta w$
 - ↳ Hinzufügen der Lernrate: $\Delta w = -\eta \nabla J(w)$
- große Lernrate ist schlecht, da das Minimum nicht gefunden werden kann

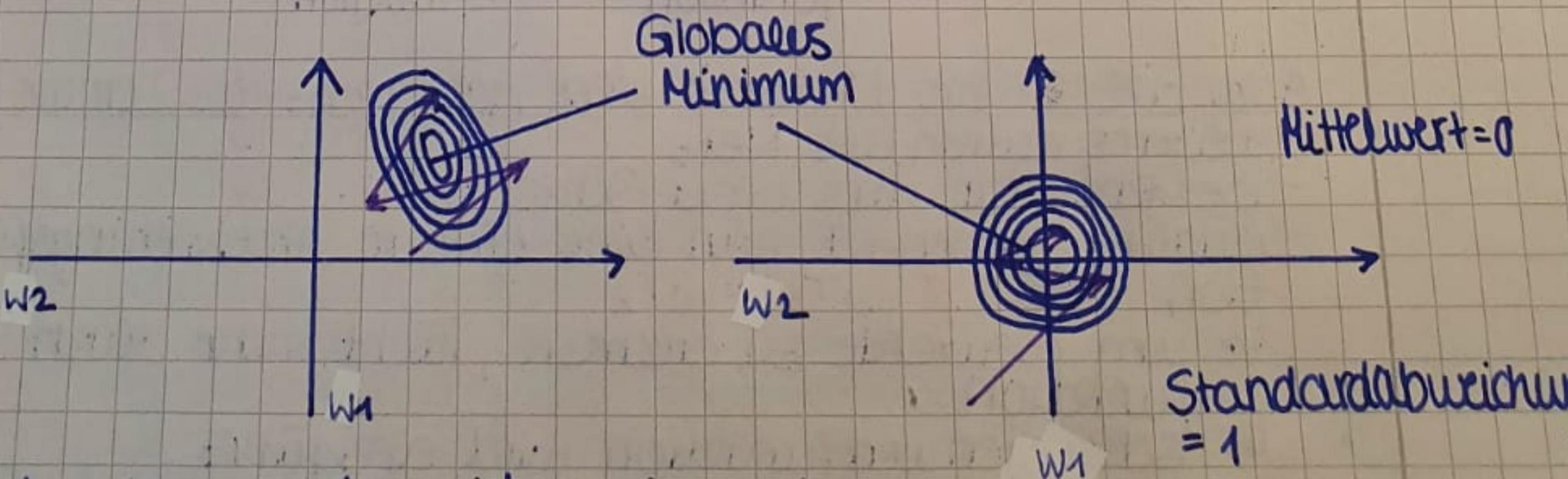


- Verfahren gelingt nicht gut, wenn die Größenordnung der Merkmale unterschiedlich ist
 - ↳ Lösung: Merkmale standardisieren

Merkmal Standardisierung

- Verschiebung der Merkmale auf Mittelwert 0 und die Standardabweichung ist 1

$$x'_j = \frac{x_j - \mu_j}{\sigma_j}$$



- ist rechnungs- und specheraufwendig
 - ↳ Lösung: Stochastisches Gradientenverfahren

Stochastisches Gradientenverfahren

- iteratives Gradientenabstiegsverfahren, Online-Gradientenverfahren

- Gewichte werden inkrementell für jedes einzelne Objekt aktualisiert: $\Delta w = \eta (y^{(i)} - \phi(w^T x^{(i)})) x^{(i)}$
- ist die Näherung des normalen Gradientenabstiegsverfahrens
- konvergiert besser, da Gradient immer neu berechnet wird (Rauschen entsteht)
- stellt Daten in zufälligen Reihenfolge zur Verfügung
 - + neue Daten lassen sich im alten System trainieren
 - + Trainingsdaten lassen sich aus Speicher löschen, bei zu niedriger Speicherkapazität
- dynamische Anpassung einer adaptiven Lernrate mit 2 Konstanten

$$\eta = \frac{C_1}{N_{\text{Epochen}} + C_2}$$

- Mini-Batch-Learning
 - ↪ zwischen normalen Batch- und stochastischem Verfahren
 - ↪ Training auf kleinere Teilmengen (schnelle Konvergenz)
 - ↪ paralleler Verarbeitung lässt vektorisierte Operationen einsetzen (bessere Performance auf der Hardware)

Implementierung Stochastisches Gradientenverfahren

- fit: Aktualisierung der Gewichte nach jedem Objekt
- partial-fit: für Online-Learning ohne Neuintialisierung der Gewichte
- shuffle: für das Durchmischen der Trainingsobjekte in jeder Epoche

Klassifizierung mit scikit-learn

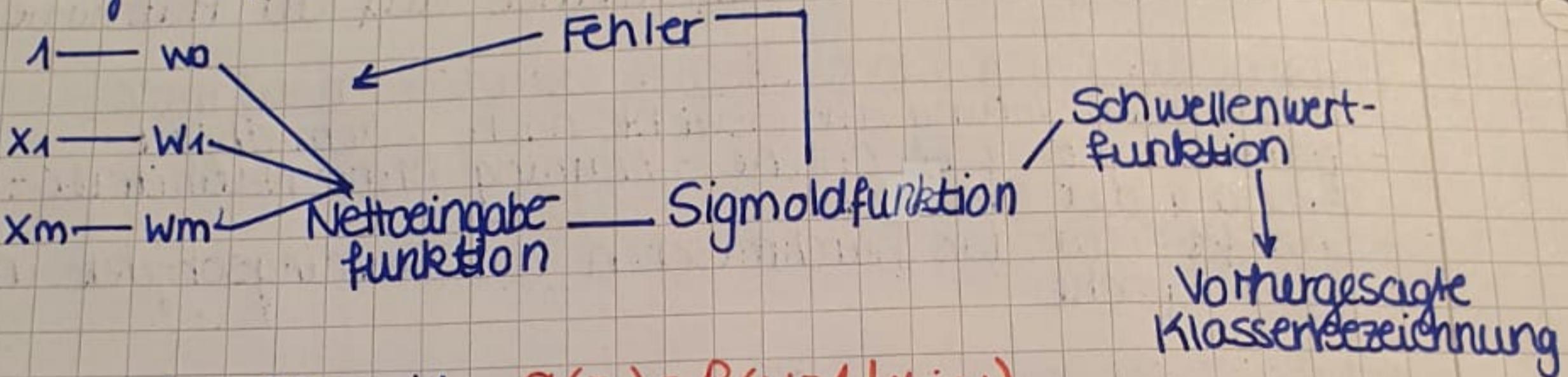
- Auswahl der Klassifizierungsalgorithmen
 - ↪ gibt keinen Klassifizierer, der für alle denkbaren Scenarien geeignet ist
 - ↪ hat bestimmte Eigenschaften und geht von bestimmten Annahmen aus
- Pragmatischer Ansatz
 - ↪ Lernalgorithmen miteinander vergleichen und das beste Modell wählen
 - ↪ Abhängigkeit der Auswahl: Anzahl der Merkmale, "Rauschen" in der Datensammlung, lineare Trennbarkeit der Klassen
- Daten sind das Wichtigste: Vorhersage und Leistung ist davon abhängig
- Allgemeines Vorgehen:
 1. Auswahl der Merkmale und Sammeln der Trainingsdaten
 2. Kriterien für die Leistungsbewertung festlegen
 3. Auswahl eines Klassifizierers und eines Optimierungsalgorithmus
 4. Bewertung der Leistung des Modells
 5. Feinbestimmung des Algorithmus
- unique() wandelt Text in ganzzahlige Werte um, da Performance besser ist als bei Strings
- train-test-split durchmischt die Werte
- Standardisierung der Merkmale (Mittelwert = 0), (Standardabweichung = 1)
- Auswahl und Anwendung des Algorithmus
 - ↪ Perzentile z.B.

- Gewichtung der Ergebnisse ...
daten
 - ↳ Anzahl der Feherklassifizierung ausgeben durch
Korrekturklassifizierung = 1 - Fehlklassifizierung
 - Achtung Überanpassung beim Trennen von Trainings- und Testdaten

Logistische Regression

- ist ein Klassifizierungsmodell
- basiert auf einem Wahrscheinlichkeitsmodell
- p : Wahrscheinlichkeit für das Eintreten eines bestimmten Ereignisses
- Chancenverhältnis für das Ereignis: $\frac{p}{1-p}$
- logit - Funktion: $\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$

- ↳ nimmt Daten zwischen 0 und 1 entgegen
- logistische Funktion:
↳ Sigmoidfunktion: $\phi(z) = \phi(w^T x) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-w^T x}}$



- Ausgabe Sigmoid: $\phi(z) = P(y=1|x; w)$
↳ z.B. Blume ergibt $\phi(z) = 0,8$ bei Wahrscheinlichkeit einer versicolor mit 80% zu sein, für eine setosa dann $P(y=0|x; w) = 1 - P(y=1|x; w) = 0,2 = 20\%$
- Schwellenwertfunktion:

$$\hat{y} = \begin{cases} 1 & \text{wenn } \phi(z) = 0,5 \\ 0 & \text{sonst} \end{cases}$$

- ⊕ nicht nur Klassifizierung, sondern auch Angabe der Wahrscheinlichkeit möglich
- ↳ vor allem in der Medizin
- Ermittlung der Gewichte: $L(w) = P(y|x; w) = \prod_{i=1}^n P(y^{(i)}|x^{(i)}; w) = \prod_{i=1}^n (\phi(z^{(i)})^{y^{(i)}} (1 - \phi(z^{(i)}))^{1-y^{(i)}})$ Likelihood-Funktion

↳ Objekte sind voneinander unabhängig
↳ Log - Likelihood ist einfacher zu berechnen / Straffunktion:

$$J(w) = -L(w) = \sum_{i=0}^n -y^{(i)} \log(\phi(z^{(i)})) - (1-y^{(i)}) \log(1-\phi(z^{(i)}))$$

↳ Straffunktion = 0, ist erfolgreich

- Gradientenabstiegsverfahren: $\frac{\partial}{\partial w_j} L(w)$

- Vorgehen:

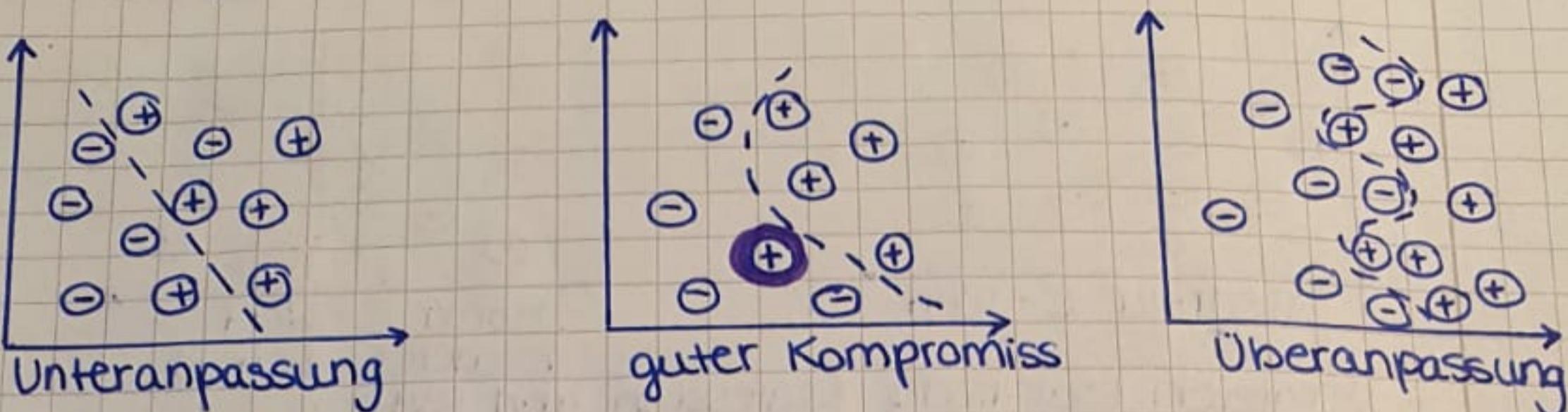
1. Adeline - Implementierung
2. Aktivierungsfunktion durch Sigmoidfunktion
3. Schwellenwertfunktion -1 oder 1 durch 0 oder 1
4. Straffunktion ersetzen

- beste Wert ist die größte Wahrscheinlichkeit

Überanpassung

- häufigste Problem beim Machine Learning

- Modell ist gut bei Trainingsdaten, bei unbekannten (Test-)daten ↳ Grund: Verallgemeinerung von Trainingsdaten ist nicht gelungen
- Überanpassung == große Varianz
↳ viele Parameter führen zu einem komplexen Modell
- Unteranpassung == große Bias
↳ nicht komplett genug, um Muster zu erkennen oder bei unbekannten Daten



- Varianz: Maß für die Konsistenz der Modellvorhersage
↳ keine Varianz = Objekt wird gleich vorhergesagt
- Bias: Maß für den systematischen Fehler, der nicht durch Zufälligkeiten geschuldet ist

Regularisierung

- Methode um Korrelationen zwischen Merkmalen handzuhaben, Rauschen aus den Daten herauszufiltern, Überanpassung zu verhindern
- Idee: Informationen zusätzlich einführen, die externe Gewichte bestrafen
↳ L^2 -Regularisierung: weiterer Term der Straffunktion

$$\frac{\lambda}{2} \|w\|^2 = \frac{\lambda}{2} \sum_{j=1}^m w_j^2 \quad \lambda = \text{Regularisierungsparameter}$$

↳ Straffunktion:

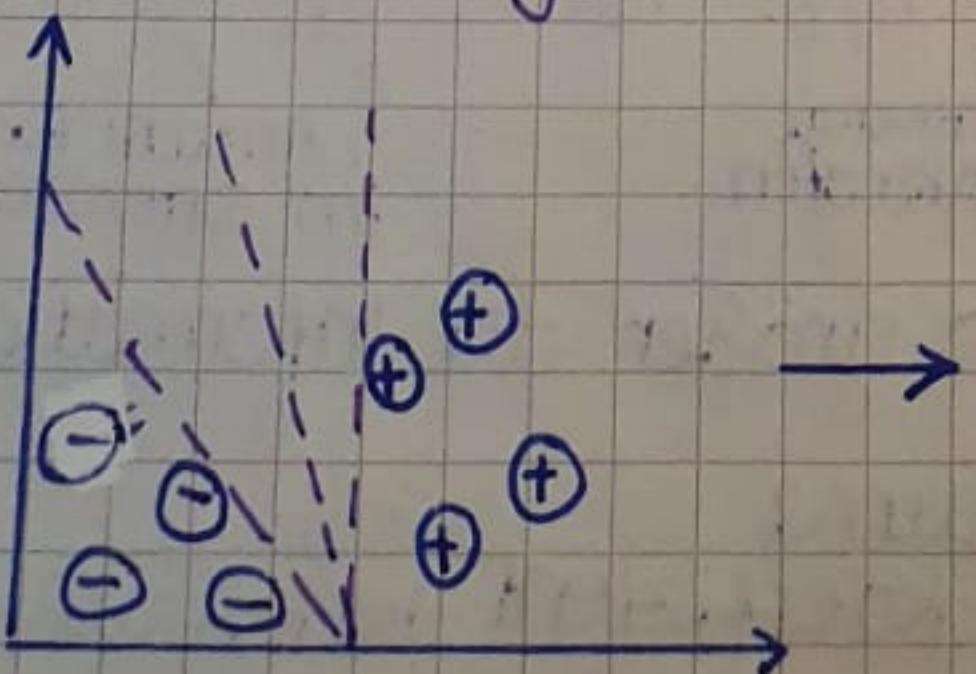
$$J(w) = \left[\sum_{i=1}^n y^{(i)} (-\log(\phi(z^{(i)}))) + (1 - y^{(i)}) (-\log(1 - \phi(z^{(i)}))) \right] + \left[\frac{\lambda}{2} \sum_{j=1}^m w_j^2 \right]$$

↳ $\lambda = 0$ strenge Anpassung, $\lambda \gg 0$ starke Regularisierung

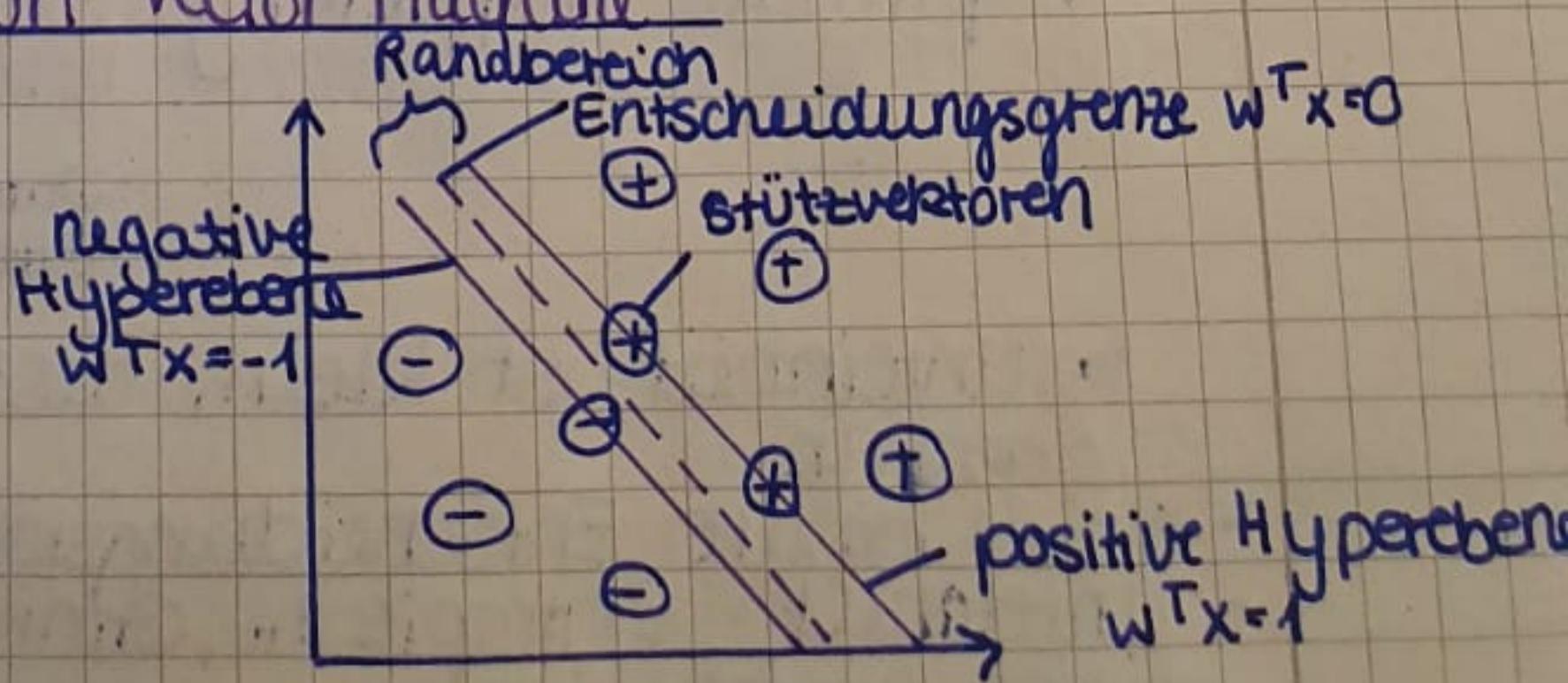
↳ Parameter C ist der Kennwert der Regularisierung

$$C = \frac{1}{\lambda} \quad C < , \text{ dann starke Regularisierung}$$

Klassifizierung mit Support Vector Machine



Welche Hyperebene?



SVM / Maximierung des Randbereichs

- Klassifizierungsfehler: $I_E(t) = 1 - \max(0, 1/t)$

$$\hookrightarrow w^T(x_{\text{pos}} - x_{\text{neg}}) = 2 \cdot \sum_{j=1}^m w_j^2 \Rightarrow \frac{2}{\|w\|}$$

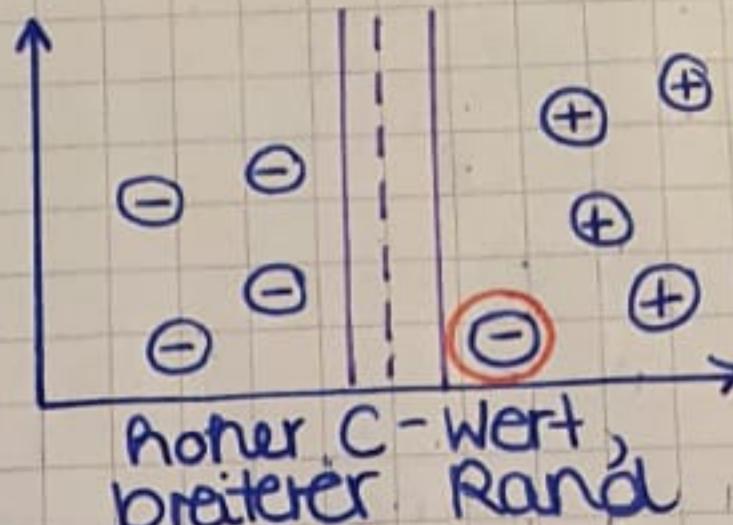
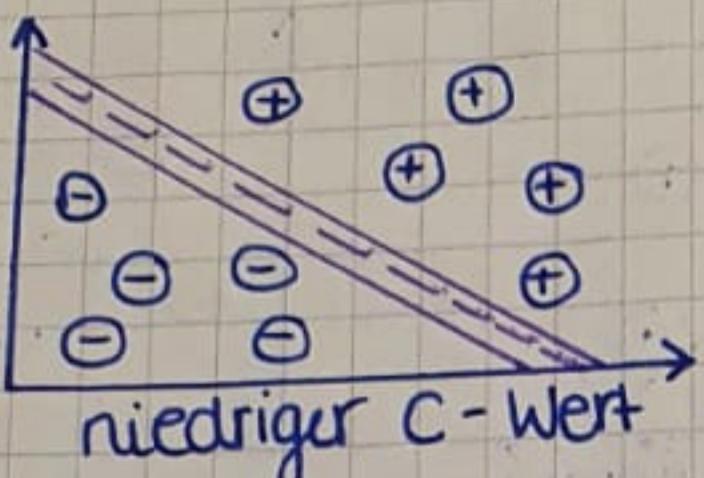
- bei nicht linear trennbaren Fällen

↪ Schlußvariable

↪ lassen Fehlklassifikationen zu

↪ lineare Beschränkungen erlaubt

$$\frac{\|w\|^2}{2} + C(\sum \xi^{(i)})$$



- vergleicht logistische Regression und SVM:

↪ ähnliche Ergebnisse

↪ logistische R. hat mehr Ausreißer in den Daten

↪ SVM sind die Stützvektoren ausschlaggebend

↪ logistische Regression ist das einfache Modell und einfacher zu aktualisieren (Vorteile bei Datenströme)

Nichtlineare Aufgabe für SVM

- nichtlinear-trennbare Daten werden mit ϕ transformiert von 2D zu 3D

$$- \text{in höherdimensionierten Raum } \phi(x_1, x_2) = (z_1, z_2, z_3) = (x_1, x_2, x_1^2, x_2^2)$$

- Rücktransformation in $-\phi$

- Ansatz: Kerneltrick, ist jedoch rechnintensiv

$$\hookrightarrow \text{Kernelfunktion: } K(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T \phi(x^{(j)})$$

$$\hookrightarrow \text{Radial Basis Funktion: } K(x^{(i)}, x^{(j)}) = \exp(-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2})$$

↪ Optimierung von γ und C

Entscheidungsbäume trainieren

- attraktives Modell zu Klassifizierung

- basiert auf "wenn, dann..."

- Wurzel beinhaltet alle Daten, Daten werden als Informationsgewinn (IG) aufgeteilt

- Aufteilung von Kindknoten bis Blattknoten

- Elemente eines Blattknotens = Klasse

- Tiefe des Entscheidungbaums muss begrenzt werden

$$\hookrightarrow \text{optimale Aufteilung: } IG(D_p, f) = I(D_p) - \sum_{j=1}^m \frac{N_j}{N_p} I(D_j)$$

Datenmenge
Elternknoten

Anzahl Exemplare
im Elternknoten

Merkmale, wo
Aufteilung ist Maß Unreinheit, im jten Kindknoten

↪ Unreinheit ist klein, desto größer der Informationsgewinn

- meist binäre Entscheidungsbäume

- Unreinheit: Entropie I_H , Gini-Koeffizient IG , Klassifizierungsfehler I_E

$$- \text{Entropie: } I_H(t) = - \sum_{i=1}^C p(i|t) \log_2 p(i|t)$$

$$- \text{Gini-Koeffizient: } IG(t) = \sum_{i=1}^C p(i|t)(1-p(i|t)) = 1 - \sum_{i=1}^C p(i|t)^2$$

- Klassifizierungsfehler: $I_E(t) = 1 - \max(p(i|t))$
- maximale Entscheidung finden:

A: $(40, 40)$

$(30, 10)$

$(10, 30)$

B: $(40, 40)$

$(20, 40)$

$(20, 0)$

$$IG_1(D_p, f) = I(D_p) - \frac{N_{links}}{N_p} I(\text{links}) - \frac{N_{rechts}}{N_p} I(\text{rechts})$$

- A: Wurzelknoten: $I_E(D_p) = 1 - 0,5 = 0,5$
 linker Knoten: $I_E(\text{links}) = 1 - 0,75 = 0,25$
 rechter Knoten: $I_E(\text{rechts}) = 1 - 0,75 = 0,25$
 $IG_1 = 0,5 - \frac{1}{2} \cdot 0,25 - \frac{1}{2} \cdot 0,25 = 0,25$

→ 40 von 80 in einem Knoten

- ↪ auch \log_2 möglich!, damit Informationsgewinn besser beobachtbar
- Tiefe muss bestimmt werden, da sonst Gefahr der Überanpassung
 ↪ da man Baum nicht nachträglich stützen kann

Random Forest

- Informationsansatz basierend
- Ensemble von Entscheidungsbäumen
 - ↪ mehrere Entscheidungsbäume mit großer Varianz kombinieren zu einem stabilen Baum
 - ↪ weniger Überanpassung
- Ansatz:
 - ↪ zufällige Stichprobe n aus Trainingsdaten (mit Ersetzung / Urne mit Zurücklegen) erstellen
 2. Entscheidungsbauum konstruieren
 - ↪ d Merkmale auswählen und aufteilen
 3. Schritt 1. und 2. k-mal wiederholen
 4. Vorhersage der einzelnen Bäume durch Mehrheitsentscheidung zusammenfassen

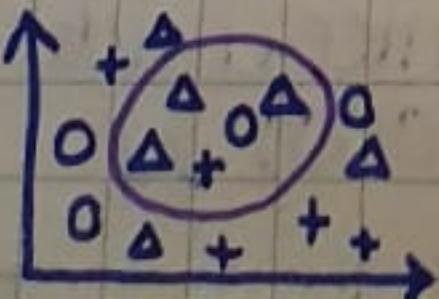
- ⊕ keine Graphfunktion möglich
- ⊕ Auswahl geeigneter Hyperparameter ist unempfindlich
- ⊕ k ist relevanter Hyperparameter

Parametrisierte Kontra nichtparametrisierte Modelle

- parametrisierte Modelle: anhand von Trainingsdaten werden Parameter abgeschätzt
 - ↪ Diskriminanzfunktion wird hergeleitet
 - ↪ kann neue Daten klassifizieren
- nichtparametrisierte Modelle: lassen sich nicht durch festen Satz von Parametern beschreiben
 - ↪ Anzahl der Parameter nimmt mit Trainingsdaten zu

k - Nearest Neighbors

- Lazy - Learning - Algorithmus / nichtparametrisiertes Modell
- Trainingsdaten werden gespeichert ohne Aufwand
- Auswahl der k nächsten Nachbarn mit deren Klassen durch die Mehrheitsentscheidung



Vorhersage: Δ

- μ beeinflusst die Überanpassung
- Minkowski-Metrik
- vorherige Standardisierung
- + Klassifizierer passt sich bei neuen Daten sofort an
- Rechenaufwand und Speicherbedarf steigt, da verwerten der Trainingsdaten nicht möglich

$$d(x^{(i)}, x^{(j)}) = \sqrt{\sum_k |x_k^{(i)} - x_k^{(j)}|^p}$$

Datenvorbereitung

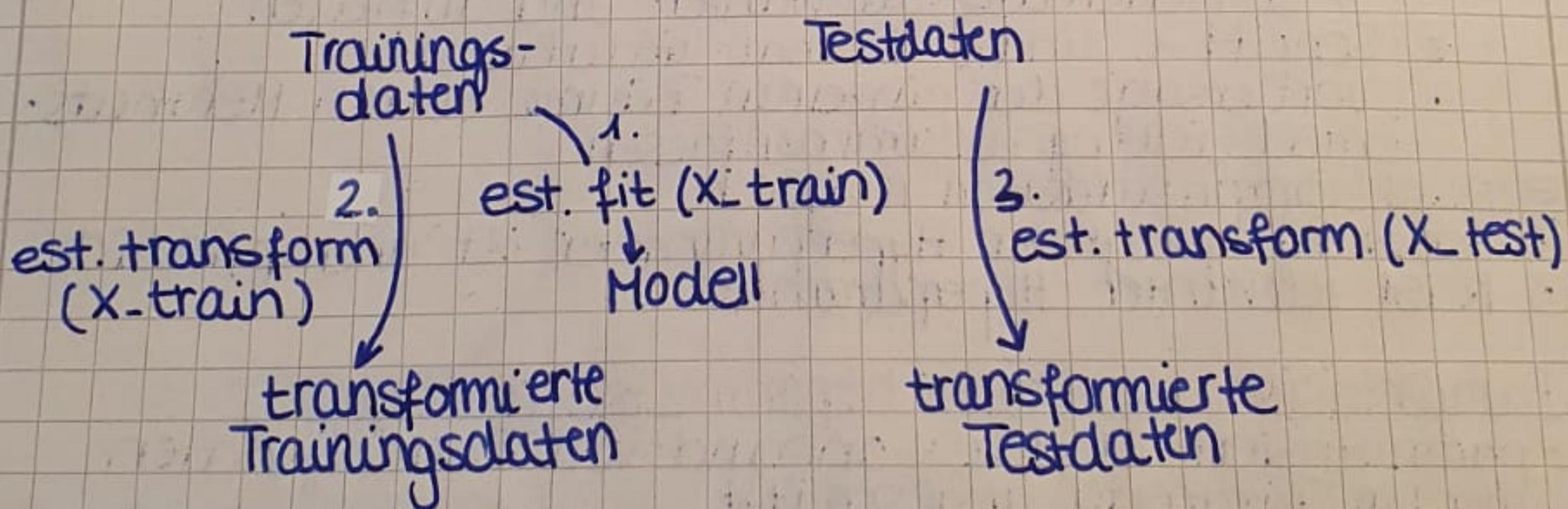
- Daten mit großem Umfang, Auswahl und hoher Qualität
- Untersuchung der Daten vor einem ML-Algorithmus
 - ↳ unvollständige und falsche Daten identifizieren
 - ↳ Aufbewahrung von Kategorien der Daten (Strings w, m, d in int-Werte)
 - ↳ Aufteilung der Datenmenge für Training und Test
 - ↳ Anpassung der Datenmerkmale

Umgang mit fehlenden Daten

- fehlende oder falsche Werte
- wenn darauf nicht geachtet wird, dann komische Werte
- isnull(): wo in einer DataFrame-Tabelle fehlen und sum(): wie viele es pro Spalte sind
- Datenmenge wird reduziert
 - ↳ evtl. reizende vernünftige Analyse mehr möglich

Fehlende Werte ergänzen

- Interpolationen, indem Daten aus anderen Werten abgeschätzt wird
 - ↳ SimpleImputer: mean, median, most_frequent, constant
- ist Teil der Transformerklasse scikit-learn



Handhabung kategorialer Daten

- nominal: bezeichnete Werte (z.B. Farbe des Shirts)
- ordinale: geordnete Werte (z.B. Größe der Shirts)
 - ↳ Konvertierung in numerische Werte
- Klassenbezeichner als Integer-Array durch LabelEncoder
- nominal wird One-Hot-Codierung verwendet
 - ↳ Alternative: pandas, kann hohe Korrelation entstehen
 - ↳ Reduzierung durch Entfernung einer Spalte

Aufteilung einer Datensammlung

- Testdatenset ist der letzte Test vor der Praxis

Anpassung der Merkmale

- Wertebereiche sollten sich nicht so sehr unterscheiden (Merkmals A: 10 - 100, Merkmals B: 1000 - 10000 \$)

- Normierung auf das Intervall $[0, 1]$: $x_{\text{norm}}^{(i)} = \frac{x^{(i)} - x_{\min}}{x_{\max} - x_{\min}}$
- Standardisierung: kann mit Ausreißer besser umgehen $x_{\text{norm}}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}$ μ_x = Mittelwert Merkmalspalte
- Standardskaler nur einmal an Trainingsdaten anwenden (fit- transform)
- mit transform mit gewonnenen Parameter auf Testdaten anwenden

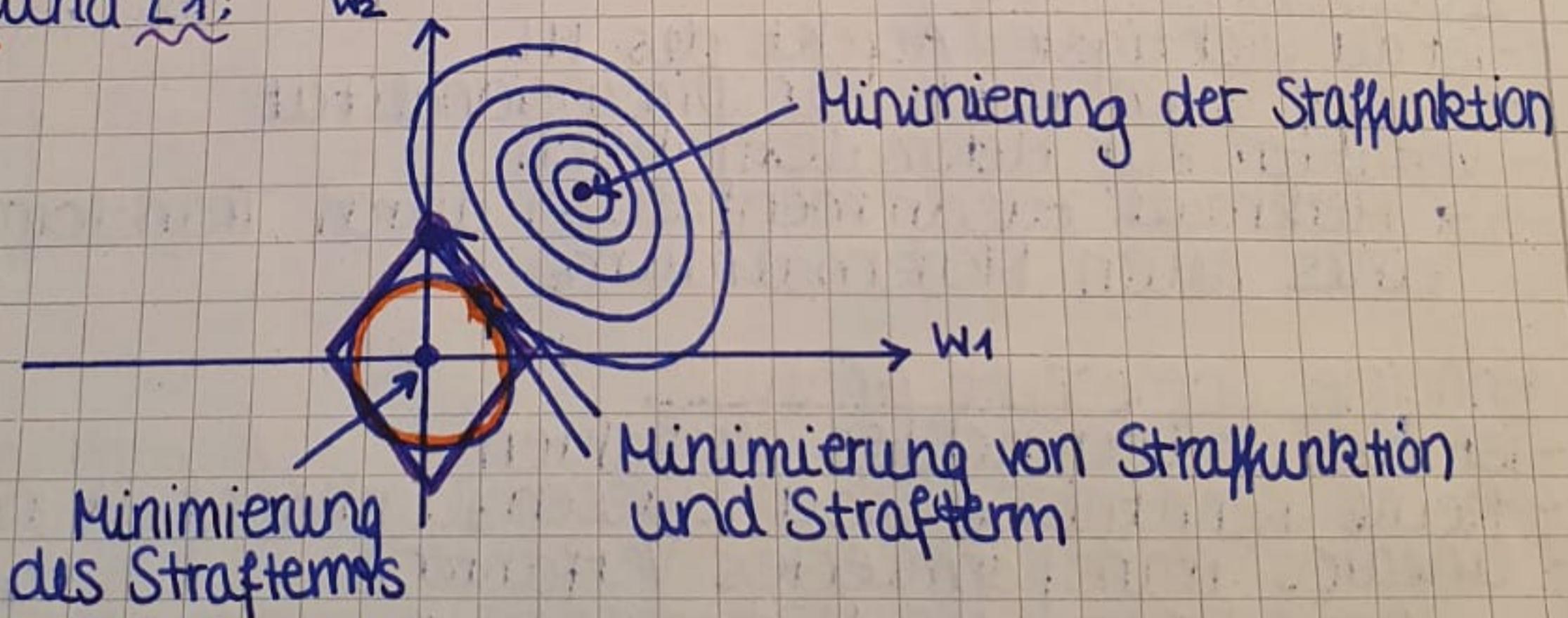
Auswahl aussagekräftiger Merkmale

- Modell funktioniert mit Trainingsdaten deutlich besser als mit den Testdaten (= Überanpassung)
 - ↪ Modell hat keine Verallgemeinerungsfähigkeit
 - ↪ Modell hat eine hohe Varianz
 - ↪ zu komplexes Modell
- Anzahl der Merkmale reduzieren = einfacheres Modell
 - ↪ Lösungen:
 - ↪ mehr Trainingsdaten sammeln: nicht immer machbar
 - ↪ Einführung Straffunktion als Regulierung
 - ↪ weniger Parameter auswählen
 - ↪ Dimensionsreduktion der Daten

Regularisierung - L_2 und L_1

- Straffunktion, um große Gewichte zu bestrafen

- $J(w) + \frac{\lambda}{2} \|w\|^2$
- L_2 -Norm: $\|w\|^2 = \sum_{j=1}^m w_j^2$
- L_1 -Norm: $\|w\| = \sum_{j=1}^m |w_j|$ mit Absolutwerten
- L_2 und L_1 :



$$- L_1: \|w\| = |w_0| + |w_1| = \begin{cases} -w_0 + |w_1|, & \text{falls } w_0 < 0 \\ w_0 + |w_1|, & \text{falls } w_0 > 0 \end{cases} =$$

$$\begin{cases} -w_0 - w_1, & \text{falls } w_0 < 0, w_1 < 0 \\ -w_0 + w_1, & \text{falls } w_0 < 0, w_1 > 0 \\ w_0 - w_1, & \text{falls } w_0 > 0, w_1 < 0 \\ w_0 + w_1, & \text{falls } w_0 > 0, w_1 > 0 \end{cases}$$

- ↪ dünn besetzte Matrizen werden begünstigt
- ↪ müssen Merkmal-Gewichte sind 0
- ↪ für hochdimensionale Datensätze

Algorithmen zur sequentiellen Auswahl von Merkmalen

- für die Dimensionsreduzierung
- 2 Hauptkategorien: Auswahl und Extrahierung von Merkmalen
- SBS Sequential Backward Selection
 - ↳ Berechnung effizienter machen, weniger Leistungseinbuße
 - ↳ gehört zum "greedy" Algorithmus
 - ↳ Entfernung eines Merkmals mit geringer Klassifizierungsrate, wiederholen
 - ↳ lokale Optimierung reicht
- Ziele:
 - ↳ d-dimensionalen Raum in k-dimensionalen Raum reduzieren
 - ↳ automatische Auswahl der Teilmenge der Merkmale mit größter Bedeutung
 - ↳ Verallgemeinerungsfehler durch Entfernen irrelevanter Merkmale verringern
 - ↳ Vorhersagekraft bei Überanpassung verbessern
- minimierende Strafpunktion für die Entfernung der Merkmale definieren
 - ↳ Festlegen des Merkmals: $x^- = \arg \max J(x_k, -x)$
 - ↳ Entfernung des Merkmals: $x_{k-1} = x_k - x^-; k := k-1$

Bedeutung von Merkmalen im Random Forest

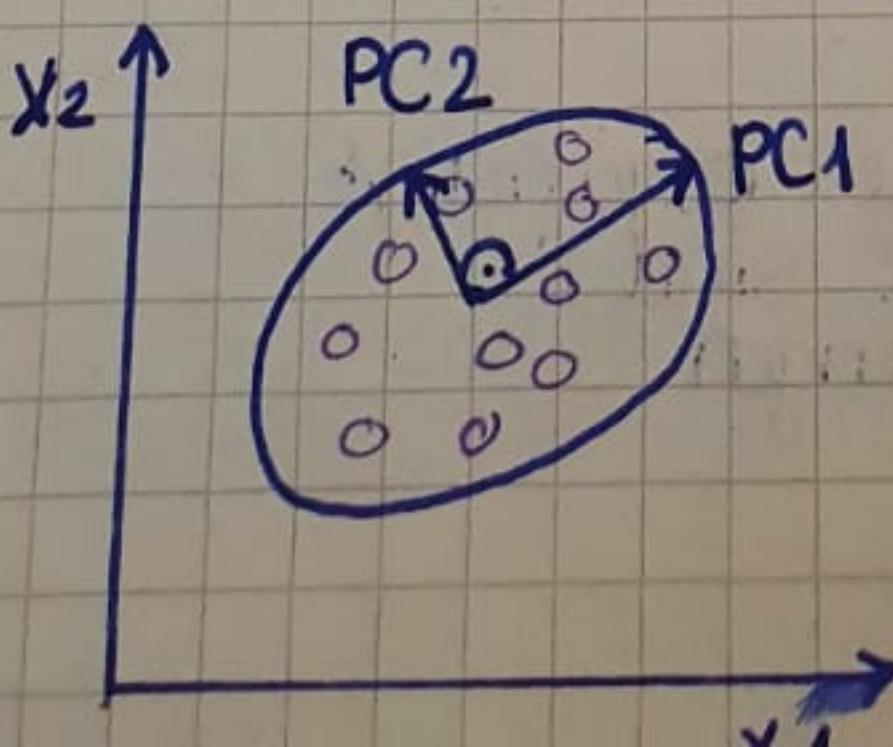
- erkennen relevanter Merkmale im Random Forest
- scikit-learn hat eine Implementierung RandomForestClassifier mit den Attributen feature_importance
- Methoden zur Merkmalreduzierung: L1 Regularisierung, SBS Algorithmus, Random Forest, SelectFromModel

Datenkomprimierung

- ist der wichtigste Aspekt des ML
- Kampf gegen Fluch der Dimensionalität
- Verfahren zur Merkmalsextraktion
 - ↳ Merkmale zusammenfassen durch Transformation eines neuen Merkmalraums

Principle Component Analysis

- ist ein unüberwachtes Verfahren
- keine Kenntnisse über Klassenzugehörigkeit notwendig
- lineares transformiertes Verfahren
 - ↳ Veränderung des Merkmalsraums und der Dimension
- Anwendung: Dimensionsreduktion, explorative Datenanalysen, Reduktion von Rauschen
- Muster zwischen Merkmalen durch Untersuchung der Korrelationen
- suchen die Richtungen maximaler Varianz



- PC1 hat eine größere Varianz

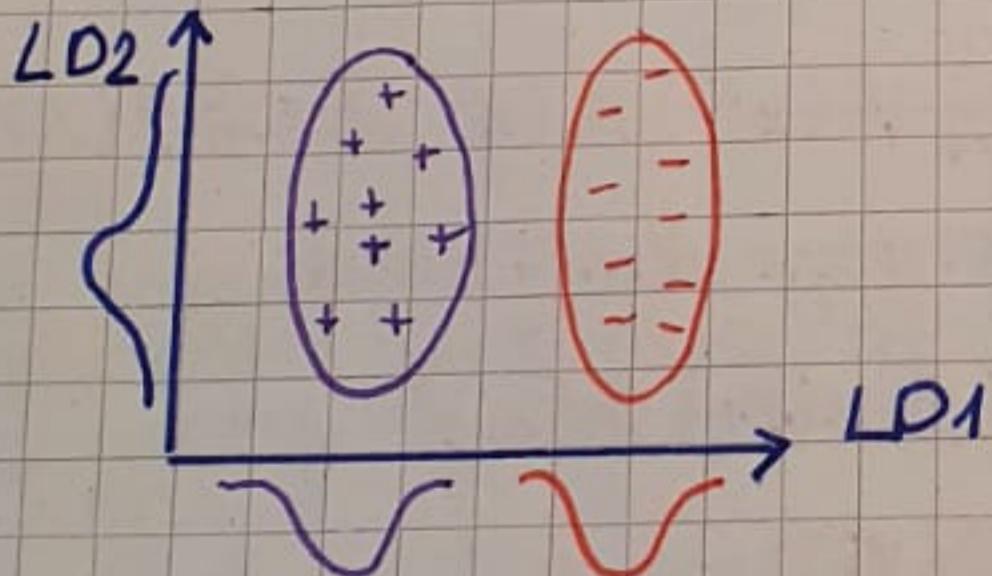
- Algorithmus:
 1. d-dimensionale Datenmenge standardisieren: $M = 0$
 2. Kovarianzmatrix konstruieren: $\sigma_{jk} = \frac{1}{n} \sum_{i=1}^n (x_j^{(i)} - \mu_j)(x_k^{(i)} - \mu_k)$
 3. Kovarianzmatrix in Eigenwerte und Eigenvektoren zerlegen
 4. Eigenwerte absteigend sortieren, um die Rangliste der Eigenvektoren zu erhalten
 5. k Eigenvektoren auswählen ($k \leq d$)
 6. Projektionsmatrix W aus k bestimmen
 7. Transformation der d-Eingabemenge X mit W im k -Merkmalsraum

↳ positive Kovarianz: Werte beider Merkmale sinken / steigen gemeinsam
 ↳ negative Kovarianz: hohe Werte eines Merkmals gehen mit niedrigen Werten des anderen Merkmals einher
 ↳ Eigenwert ist λ -Skalar
 ↳ Matrixmultiplikation mit einem Eigenvektor ergibt ein Vielfaches λ des Eigenvektors: $\Sigma v = \lambda v$
 ↳ Anteile der Eigenwerte an der Varianzaufklärung:

$$\frac{\lambda_j}{\sum_{j=1}^d \lambda_j}$$

Lineare Diskriminanzanalyse

- für die Merkmalsextraktion 1-auswahl, für die Reduzierung der Dimension
- Erhöhung der Effizienz der Berechnung
- nicht regularisierbare Modelle kann die Überanpassung verringert werden
- Merkmalsraum finden, der die Trennbarkeit der Klassen optimiert
- überwachtes Verfahren



- Voraussetzungen:

- ↳ Daten sind normalverteilt
- ↳ Kovarianzmatrizen der Klassen sind identisch
- ↳ Merkmale sind statistisch voneinander unabhängig

- Algorithmus:

1. d-dimensionale Datenmenge standardisieren
2. Berechnung des Mittelwertsvektors jeder Klasse: $m_i = \frac{1}{n_i} \sum_{x \in D_i} x$
3. 2 Streumatrizen konstruieren
 - ↳ S_B : Streuung zwischen den Klassen
 - ↳ S_W : Streuung innerhalb einer Klasse
4. Berechnung der Eigenwerte und -vektoren mit $S_W^{-1} S_B$
5. Eigenwerte absteigend sortieren - Reihenfolge Eigenvektoren
6. k-Eigenvektoren zu größten Eigenwerten auswählen und $d \times k$ -dimensionale Transformationsmatrix W konstruieren
7. Projektion der Daten auf neuen Merkmalsraum
 - ↳ Klassenzugehörigkeit wird verwendet
 - ↳ Streumatrizen: $S_W = \sum_{i=1}^c (S_i) = \sum_{i=1}^c (\sum_{x \in D_i} (x - m_i)(x - m_i)^T)$
 - ↳ $S_B = \sum_{i=1}^c n_i (m_i - m)(m_i - m)^T$

- scikit-learn mit der Klasse LinearDiscriminantAnalysis

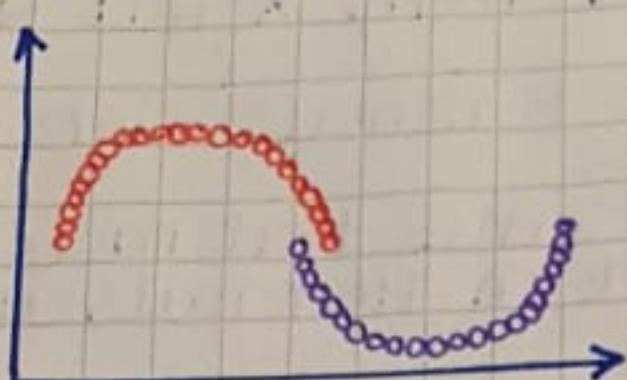
Kernel-PCA (Kernel-Hauptkomponentenanalyse) für nicht-lineare Zuordnung

- Trennung eigentlich nur, wenn linear
↳ Realität, nicht immer lineare Trennbarkeit

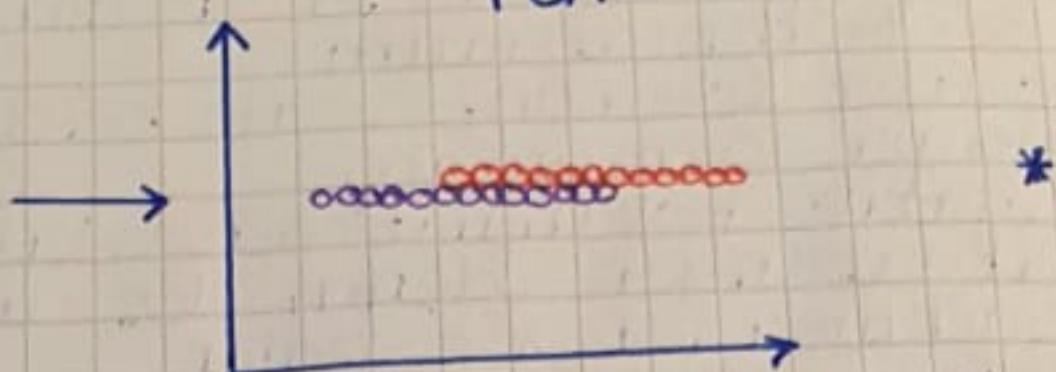
- nicht lineare trennbare Datenmengen → PCA, LDA &

↳ Kernel-PCA ↴

- Halbmondformen:



PCA:



- Ziel Kernel-PCA:

↳ Reduzierung der Dimensionen

↳ nicht trennbare Daten in neue niedrigdimensionale Unterräume transformieren

$x^{(1)}, x^{(2)}, \dots, x^{(n)} \in \mathbb{R}^m$ zu $t^{(1)}, t^{(2)}, \dots, t^{(n)} \in \mathbb{R}^r$ $r < m$

⊖ Schritte sind rechnintensiv

- Algorithmus:

1. Daten aus dem Eingaberaum in höher dimensionalen Merkmalsraum transformieren

2. PCA durchführen

3. Daten in Eingaberaum zurück projizieren

→ Skalarprodukt der Objekte aus dem ursprünglichen Merkmalsraum durch ϕ zu ersetzen: $\phi: \mathbb{R}^m \rightarrow \mathbb{P} \quad x \rightarrow x_F$

- Mittelwerte werden im Merkmalsraum um 0 zentriert:

$$\mu_F = \frac{1}{n} \sum_{k=1}^n \phi(x_k) = 0$$

- Kovarianzmatrix C : $C = \frac{1}{n} \sum_{i=1}^n \phi(x^{(i)}) \phi(x^{(i)})^T = \frac{1}{n} \phi(X)^T \phi(X)$

- Eigenwert-Gleichung: $Cv = \lambda v \dots k\alpha_j = n\lambda_j \alpha_j$

- Nebenrechnung: $v = \frac{1}{\sqrt{n}} \sum_{i=1}^n (\phi(x_i)v) \phi(x_i)^T$

↳ Eigenvektoren: $v = \sum_{i=1}^n \alpha_i \phi(x_i)$

- dass Mittelwert $\phi(x_i)$ immer 0 ist, ist nicht gültig

↳ zentrierter Merkmalsraum:

$$\tilde{\phi}(x_i) = \phi(x_i) - \frac{1}{n} \sum_{k=1}^n \phi(x_k)$$

Gebrauchliche Kernel-Funktionen

- Polynomiale Kernel mit Schwellenwert θ und Potenz p : $K(x^{(i)}, x^{(j)}) = (x^{(i)T} x^{(j)} + \theta)^p$

- Sigmoid Kernel: $K(x^{(i)}, x^{(j)}) = \tanh(\eta x^{(i)T} x^{(j)} + \theta)$

- Gauß oder Radial Basis Funktion:

$$K(x^{(i)}, x^{(j)}) = \exp\left(-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2}\right) = \exp(-\gamma \|x^{(i)} - x^{(j)}\|^2)$$

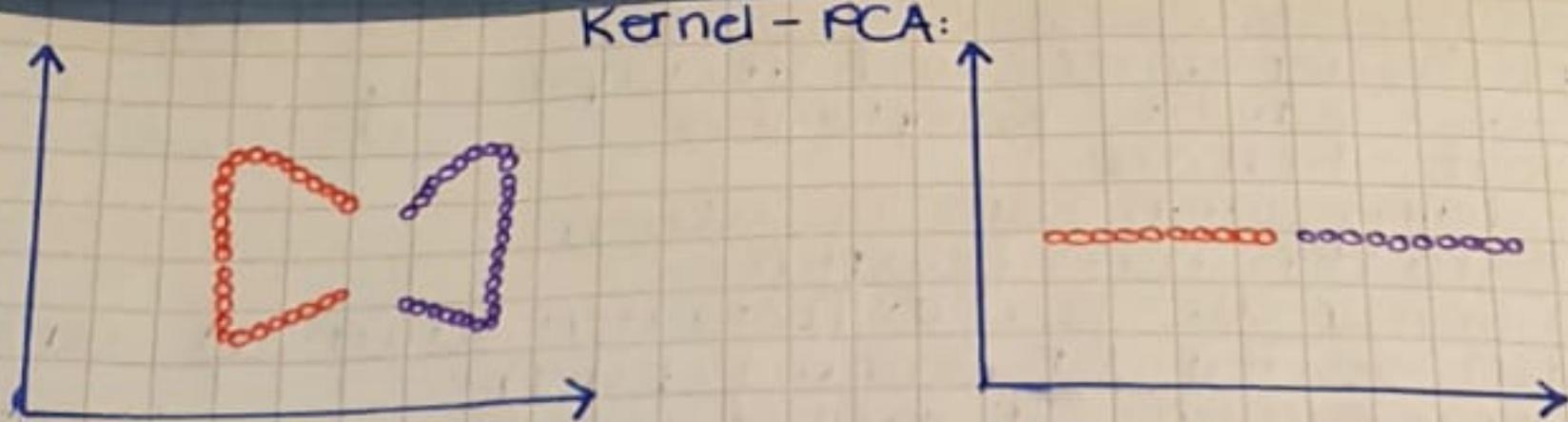
- Vorgehen:

↳ Kernel auswählen

↳ normalisierte Kernel Matrix \tilde{K} aus Daten erzeugen

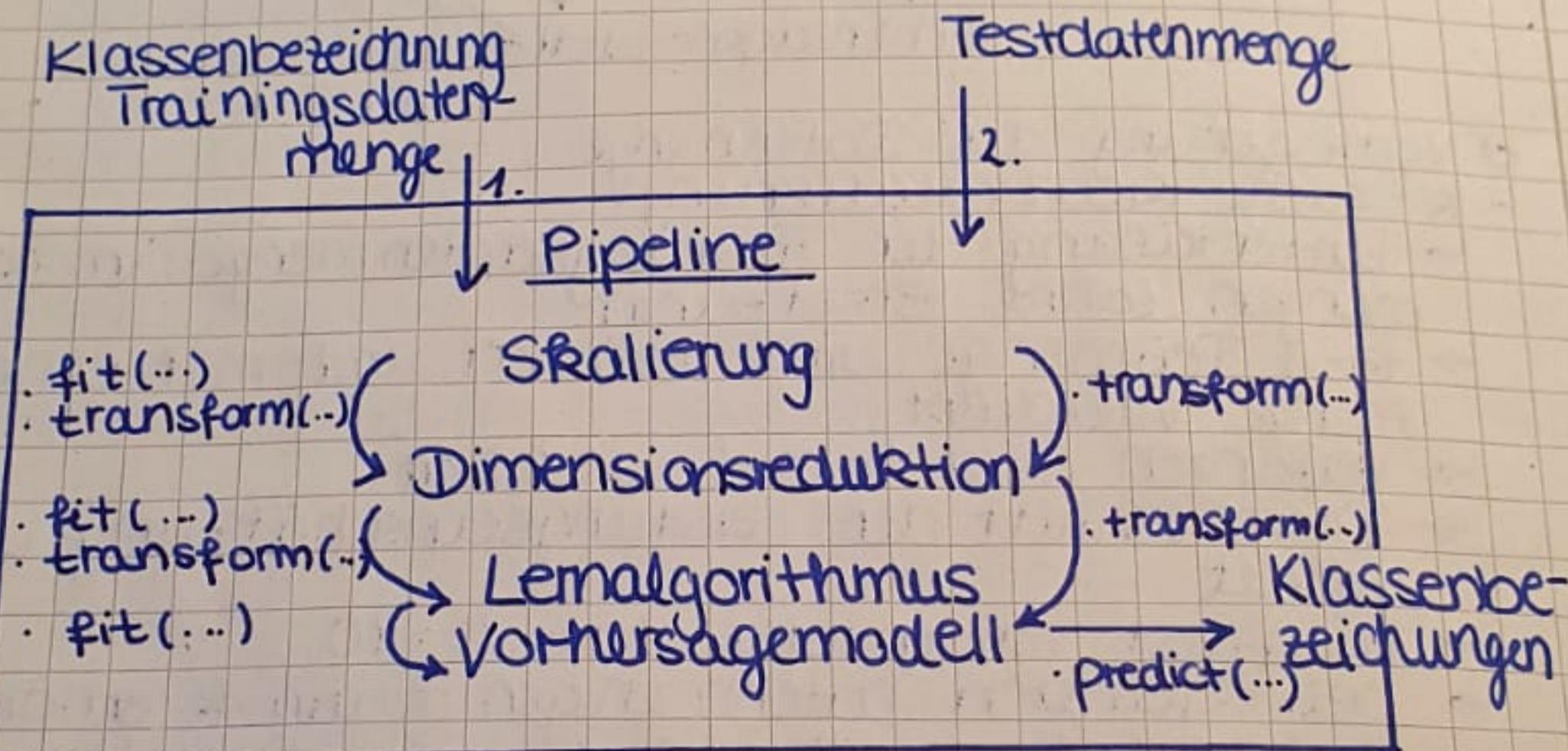
↳ Eigenwertproblem lösen

↳ Datenpunkte transformieren: $y_j = \sum_{i=1}^n \alpha_i K(x_i, x_j) \quad j = 1, 2, \dots, d$



Modellbewertung und Hyperparameterbestimmung

- viele Lernalgorithmen haben zur Klassifizierung noch zusätzliche Parameter
- Hyperparameter: Lernrate, Regularisierungs-Parameter
↳ Anzahl der Tiefe im Entscheidungsbaum
- verfahrensfehlerfreie Bewertung der Leistung eines Modells wird angestrebt
 - ↳ Bewertungskriterien anhand der Leistung
 - ↳ Feinabstimmung von Lehrmodellen
 - ↳ typische Schwierigkeiten leisten
- Pipelines bauen: Skalieren, Komprimieren



- .fit:

1. verwendet Trainingsdaten
2. Scaler wird ausgeführt
3. übergibt die transformierten skalierten Daten an den PCA
4. PCA wird ausgeführt
5. übergibt transformierten reduzierten Daten an den Schätzer
6. Schätzer wird ausgeführt

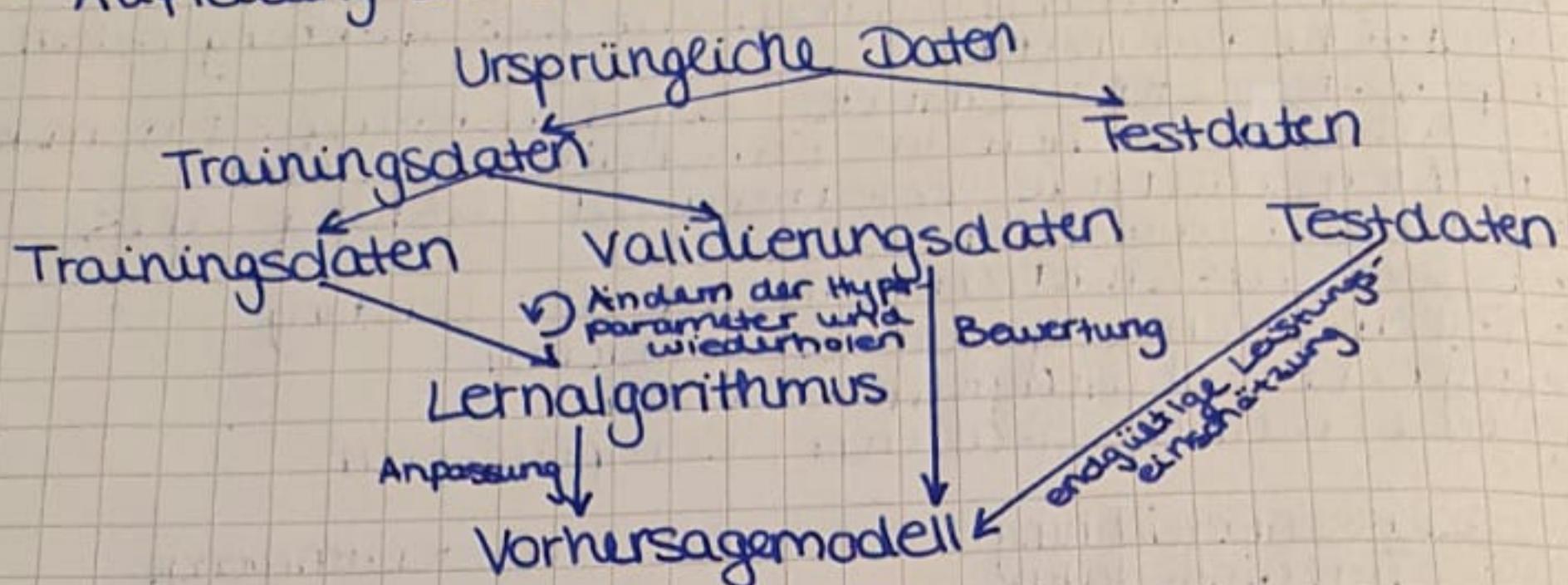
- .predict:

1. verwendet Testdaten
2. Scaler wird ausgeführt
3. übergibt transformierten skalierten Testdaten
4. PCA wird mit fit ausgeführt
5. übergibt transformierten reduzierten Daten an den Schätzer
6. Schätzer wird mit fit ausgeführt und berechnet die Vorhersage

- Pipeline vereinfacht die Modellbestimmung und -benutzung

Beurteilung des Modells

- Problem der Unter- und Überanpassung
 - ↳ Bias - Varianz - Kompromiss
- Holdout - Methode
 - ↳ Trainingsdaten zum Trainieren
 - ↳ Testdaten zum Bewerten
- wenn immer die gleichen Testdaten verwendet werden, werden sie Teil der Trainingsdaten
 - ↳ Überanpassung ist wahrscheinlich
- Aufteilung der Daten in 3 Teile:

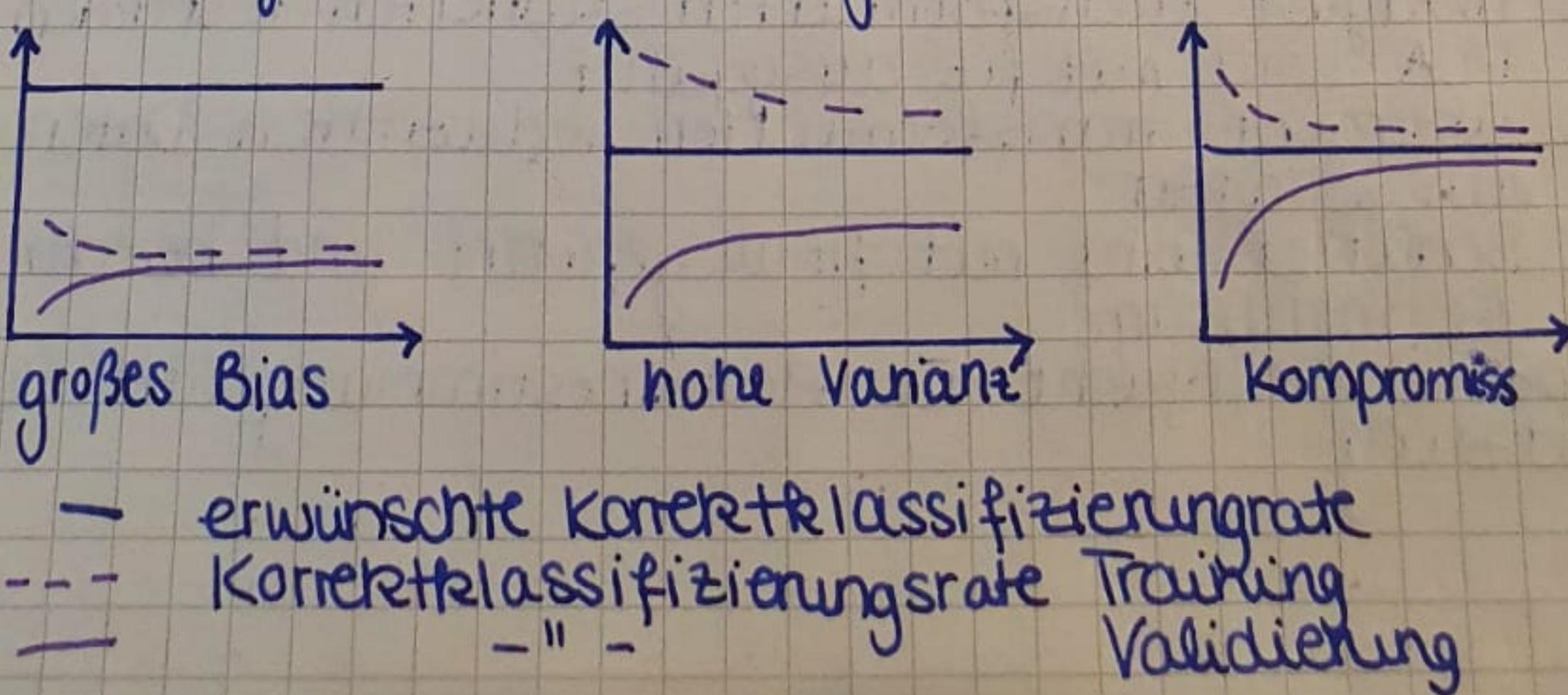


Θ Verringerung der Datenmenge

- k -fache Kreuzvalidierung:
 - ↳ Unterteilung der Trainingsdatenmenge in k Teilmengen (ohne Ersetzung)
 - ↳ $k-1$ Teilmenge wird trainiert und mit einer Teilmenge validiert
 - ↳ Verfahren wird k mal wiederholt
 - ↳ Durchschnitt der Leistungseinschätzung wird verwendet
 - ↳ häufigst verwendeter Wert: $k=10$
 - ↳ bei kleinen Datenmengen kann k erhöht werden
 - ↳ Extremfall: $k=n$ (Leave - One - Out - Kreuzvalidierung)
- Stratifizierte k -fache Kreuzvalidierung:
 - ↳ achtet auf die Verteilung der Klassen
 - ↳ Teilmenge hat genauso viele Klassen wie Trainingsdaten
 - ↳ cross_val_score

Überprüfung des Lernverlaufs

- Lernkurve: Kompromissfindung zwischen Bias und Varianz
- Validierungskurven: Umschiffung von typischen Schwierigkeiten von Lernalgorithmen



- Learning-curve für k -fach Kreuzvalidierung
- Validation-curve für Validierungskurve
→ Regularisierungsparameter C festlegen

Feinabstimmung eines Lernmodells durch Rastersuche

- 2 externe Parameter: Hyperparameter und Trainingsdaten
- Verbesserung der Leistungsfähigkeit eines Modells durch Optimierung der Kombination der Hyperparameter
- findet die ideale Parameterkombination sehr gut
- ⊖ rechenintensiv
- Randomized SearchCV
 - ↳ wie viel Rechenaufwand will man
 - ↳ zufällige Parameterkombinationen auswerten, wo macht es Sinn weiterzusuchen

Algorithmiauswahl

- R-fache Kreuzvalidierung mit Rastersuche kombinieren
- äußere Schleife:
 - ↳ Aufteilung in Trainings- und Testdatenteilmengen
 - ↳ R-fache Kreuzvalidierung
- innere Schleife:
 - ↳ zur Auswahl des Modells
 - ↳ j-fache Schleife
- = verschachtelte Kreuzvalidierung ($k \times j$)

Kriterien zur Leistungsbewertung

- Kriterien: Precision, Recall, F_1
- Wahrheitsmatrix:
↳ Konfusionsmatrix / confusion-matrix

| | | vorhergesagte Klasse | | tatsächliche Klasse |
|---|---|----------------------|----------------------|---------------------|
| | | P | N | |
| P | P | richtig positiv (TP) | falsch negativ (FN) | tatsächliche Klasse |
| | N | falsch positiv (FP) | richtig negativ (RN) | |

- falsch positiv und falsch negativ kann zu Unschärfe sorgen (sollten gegen 0 gehen)
- Korrektklassifizierungsrate:

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

- Fehlerquote:

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

- Genauigkeit:

$$GEN = prec = \frac{TP}{TP + FP}$$

- Trefferquote:

$$TQ = recall = \frac{TP}{TP + FN}$$

- F_1 -Maß:

$$F_1 = \frac{2 \cdot GEN \cdot TQ}{GEN + TQ}$$

ROC Diagramme

- = Receiver - Operating - Characteristic
- Grenzwertoptimierungskurve
- ↳ anhand FP und RP - Rate
- Algorithmus:
 - ↳ Entscheidungsgrenze des Klassifizierers verschieben
 - ↳ Raten berechnen
 - ↳ RP als Funktion der FP - Rate auftragen
- Fläche unter der Kurve - Maß für die Leistung eines Klassifizierers

Umgang mit Mehrfachklassifizierungen

- One - vs - All (OvA) - Klassifizierung
- Macro - und Micro - Methoden zur Mittelwertbildung

↳ Micro:

$$GEN_{\text{micro}} = \frac{TP_1 + TP_2 + \dots + TP_R}{TP_1 + TP_2 + \dots + TP_R + FP_1 + FP_2 + \dots + FP_R}$$

↳ Macro:

$$GEN_{\text{macro}} = \frac{R}{R}$$

↳ Micro für Gleichgewichtung einzelner Vorhersagen

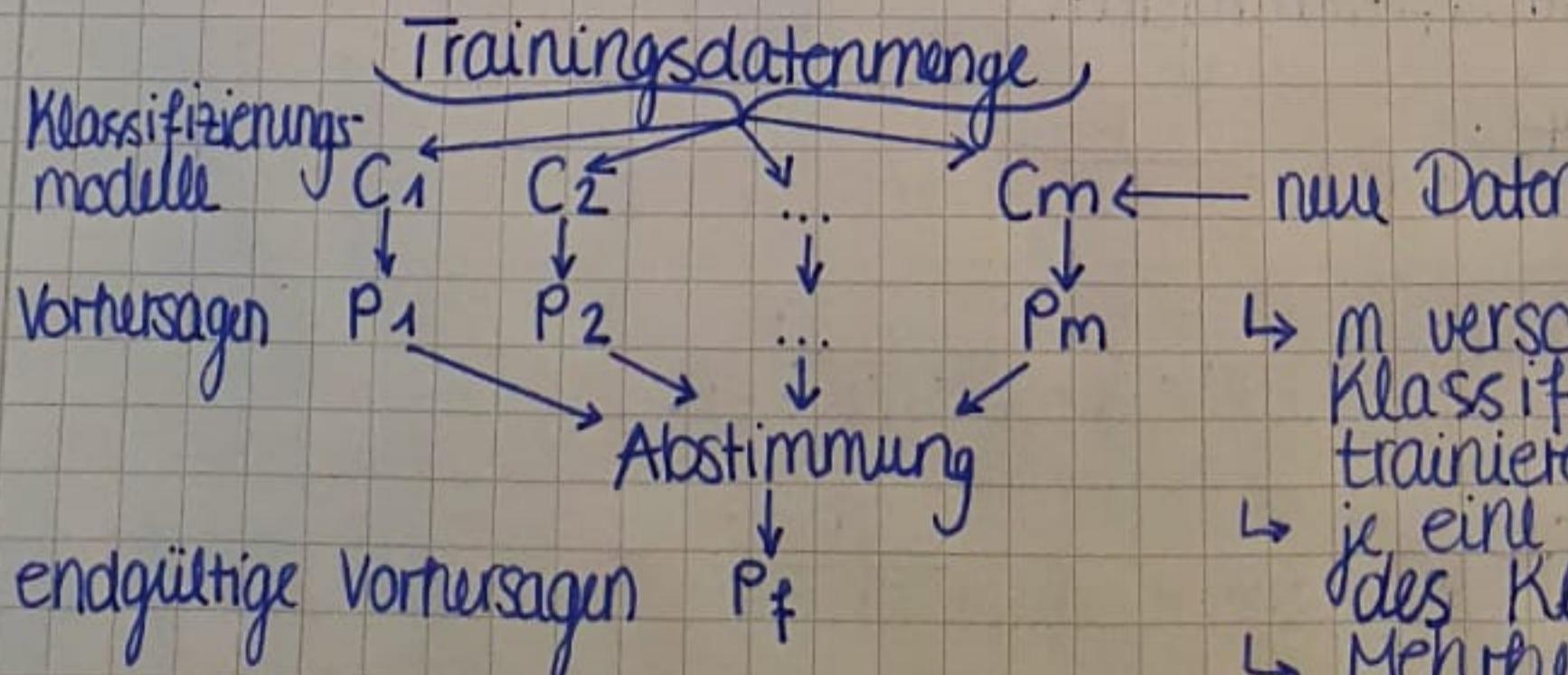
↳ Macro für häufigste Klassenbezeichner

Handhabung unausgewogener Klassenverteilungen

- Menge von Trainingsdaten, ist eine Klasse überrepräsentiert
- Uunausgewogene Klassenverteilung
 - ↳ wirkt sich auf die Bewertung des Lernmodells aus
 - ↳ auf die Anpassung des Modells (Straffunktion verschiebt sich zugunsten einer Klasse)
- Möglichkeiten
 1. Parameter class_weight = 'balanced'
 2. up - oder down - Sampling (Erhöhen oder Erniedrigen des Anteils der wenigen oder häufigen Klassen)
 - ↳ resample

Ensemble Learning

- Kooperation statt Konkurrenz
- mehrere Klassifizierer zu einem Meta - Klassifizierer
 - ↳ bessere Verallgemeinerungsfähigkeit
- Mehrheitsentscheidung
 - ↳ Einstimmigkeit 0 0 0 0 0 0
 - ↳ absolute Mehrheit 0 0 0 0 △ △
 - ↳ relative Mehrheit 0 0 0 △ △ □



- ↳ m verschiedene Klassifizierer C_m trainieren
- ↳ je eine Vorhersage des Klassifizierers
- ↳ Mehrheitsentscheidung durch Abstimmung

$\hat{y} = \text{Modalwert } \{C_1(x), \dots, C_m(x)\}$

↳ Modalwert = Wert, der am häufigsten vorkommt

→ binäre Klassifikation: Klasse 1: $C(x) = -1$, Klasse 2:
 $C(x) = 1$
 $\hat{y} = \text{sign} \left[\sum_{j=1}^m C_j x_j \right] = \begin{cases} +1 & \text{wenn } \sum_{j=1}^m C_j x_j > 0 \\ -1 & \text{sonst} \end{cases}$

- warum Ensemble?
 - ↳ da unabhängig trainiert wird
 - ↳ Fehlerquote ist kleiner, als die von einzelnen Klassifizierer
 - ↳ Wahrscheinlichkeit, dass genau e Klassifizierer einen Fehler machen: $\binom{m}{e} p_e^e (1-p_e)^{m-e}$ p_e = Fehlerquote
 - ↳ Wahrscheinlichkeiten oberhalb der Mehrheit zusammenzählen: $P_{\text{ensemble}} = \sum_{e=m/2}^m \binom{m}{e} p_e^e (1-p_e)^{m-e}$

- Algorithmus:

1. Datensatz einlesen und Klassen encodieren
 2. Aufteilung der Trainingsdatenmenge in 50:50
 3. Klassifizierer anlegen (z.B. logistische Regression, Entscheidungsbaum, ...)
 4. Pipelines anlegen für die Skalierung
 5. Labels definieren
 6. 10-fach Kreuzvalidierung
- Klassifizierer können auch gewichtet werden: $\hat{y} = \arg \max \sum_{j=1}^m w_j x_j (C_j(x)=1)$
 z.B. $w_1 = 0,2, w_2 = 0,2, w_3 = 0,6$ wenn $C_1, C_2 = 0, C_3 = 1$
 $\hat{y} = \text{Modalwert} \{0, 0, 1, 1, 1\} = 1 \Rightarrow C_3$

- Mehrheitsentscheidung mit Wahrscheinlichkeiten:

$$\hat{y} = \arg \max_i \sum_{j=1}^m w_j p_{ij}$$

Ensembles mit Stichproben

- Bagging: Bootstrapping - Aggregation
 - ↳ verwandt mit Random Forest
- Klassifizierer bekommt von Trainingsmenge zufällige Teilmengen
- jedes Objekt steht bei jeder Ziehung zur Verfügung
- Bootstrapping mit Ersetzung
- Objekt kann gar nicht oder mehrfach in Objekt sein
- + kann die Varianz eines Modells verringern
- Einsatz nur bei geringem Bias

AdaBoost - Adaptives Boosting

- 1990 Konzept von Schapire
- Trainingsdaten ohne Ersetzung
- Algorithmus:
 1. zufällige Stichprobe wählen um schwachen Klassifizierer zu trainieren C_1
 2. zweite Stichprobe, 50% der zuvor fehlerklassifizierten Objekte nehmen für schwachen Klassifizierer C_2
 3. dritte Stichprobe aus unterschiedlichen Klassifizierungen von C_1 und C_2 , um C_3 zu trainieren
 4. Kombination C_1, C_2, C_3 durch Mehrheitsentscheidung
- + Bias und Varianz können verkleinert werden
 - Gewichte werden nach jedem Durchgang neu definiert, je falsch, desto höher die Gewichte
 - gefährlich, da mehrmalige Verwendung der Trainingsdaten zu optimistischen Verallgemeinerung führt

Vorbereitung der Textanalyse

- durch eine Stimmungsanalyse
 - 1. Download der Datei
 - 2. Einlesen in ein Dataframe
 - 3. Speichern der Daten in eine Excel-Datei
- Stimmungsanalyse:
 - ↳ durch Meinungen, Bewertungen, Empfehlungen
- Natural Language Processing (NLP)
 - ↳ verarbeitet die natürliche Sprache

Bag-of-Words-Modell

- Text muss in numerische Zahlen umgewandelt werden
- Vokabular eindeutiger Tokens erstellen
- für jedes Textdokument ein Merkmalsvektor erstellen
- Raw Term Frequency
 - ↳ Vorkommenshäufigkeit
 - ↳ Vorkommen des Wortes t in einem Dokument
- Worte in Merkmal umwandeln durch CountVectorizer
- Wörter kommen in mehrere Dokumente vor
- inverse document frequency:

$$idf(t, d) = \log \frac{1 + na}{1 + df(d, t)}$$

↓
Inverse Vorkommensfrequenz

Anzahl Dokumente
mit Wort t

- tf-idf Maß: $tf - idf(t, d) = tf(t, d) \cdot idf(t, d)$ durch TfIdTransformer
 - ↳ bei Scikit-Form + 1 in Formel
- z. B. "is" kommt 3 mal vor in 3 Dokumenten
 - tf = 3 und idf = 3
 - ↳ $df("is", d_3) = \log \left(\frac{1+3}{1+3} \right) = 0$
 - tf idf() = $3 \cdot (0+1) = 3$

Textdaten bereinigen

- Daten vorverarbeitung mit Sonderzeichen und Emoticons
- Sonderzeichen enthält HTML artigen Text
 - ↳ sollte entfernt werden
- Emoticons weisen auf Stimmungen
 - ↳ erhalten bleiben, zwischenspeichern
- unerwünschte Sonderzeichen entfernen mit regulären Ausdrücken
- was keine Wörter sind entfernen durch regexp "[\W]"
- alles in Kleinschreibung umwandeln durch text.lower()
- Emoticons anhängen

Text in Token zerlegen

- Worte mit Leerzeichen trennen
 - ↳ z. B. Hallo du! → 'Hallo' 'du'
- Worte auf ihrem Stamm reduzieren
 - ↳ z. B. Läufer und laufen
 - ↳ Porter Stemming Algorithmus
- Stopwörter
 - ↳ vor allem Artikel = unnütze Informationen

Logistisches Regressionsmodell für die Dokumentklassifizierung

1. Daten in Trainings- und Testdaten trennen
2. TfIdVectorizer mit Transformeur
3. Pipeline erzeugen
4. Optimierung der Hyperparameter mit einer Gittersuche
 - ↳ probieren technik mit Stopwörtern ohne, usw.

5. Ausgabe der besten Rastersuche
6. Überprüfung des Ergebnisses für die Testdaten

Vorbereitung großer Datensätze

- Rastersuche ist rechintensiv
- Arbeitsspeicher wird ausgelastet
 - ↳ Lösung: Out-of-Core-Learning, das schrittweise Klassifizieren kleinerer Teilmengen
 - ↳ SGD Classifer
 - ↳ Dokumente werden direkt vom Laufwerk eingelesen
 - ↳ Tokenizer mit Verwendung von Stopwörtern
 - ↳ Funktion liest einzelne Zeile der CSV-Datei ein
 - ↳ Batchs der Reihe nach abarbeiten
- Achtung:
 - ↳ CountVectorizer hat nicht alle Dokumente zur Verfügung
 - ↳ TfidfVectorizer muss alle Merkmalsvektoren in den Speicher laden
 - ↳ Lösung: HashingVectorizer ist von den Daten unabhängig und verwendet den Hashing-Trick
- Daten gleichzeitig trainieren
 - ⊕ gute Vorhersage
 - ⊖ schlechte Performance
- SGD
 - ⊕ gute Trainingsperformance
 - ⊖ schlechte Vorhersage
- Serialisierung
 - ↳ Python Objektstruktur wird in Bytecodes gespeichert
 - ↳ pickle-Funktion
 - ↳ dump Stopwörter und Classifier speichern

Status

- pickle Framework lässt Modelle persistieren
- ↳ Wiederverwendung möglich

Topic Modeling mit latenter Dirichlet-Allokation

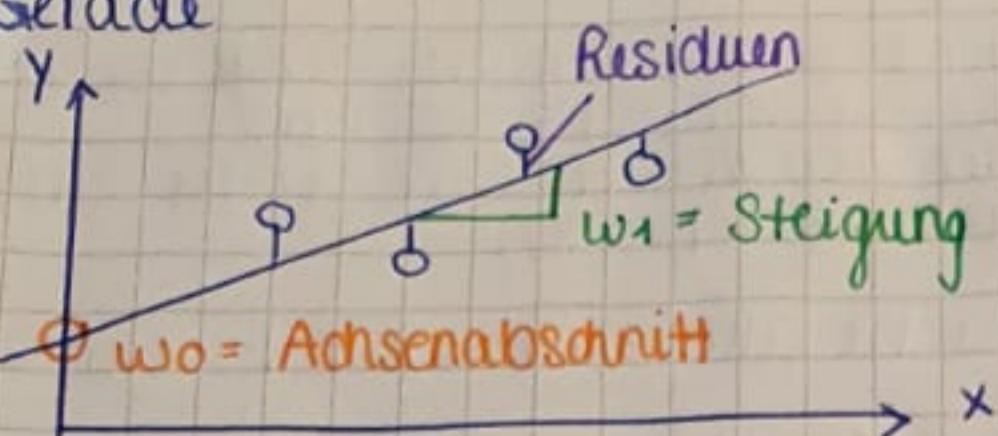
- Unüberwachtes Lernen
- ungekennzeichnete Textdokumente in Themenbereiche zuordnen
- Topic Modeling
- latenter Dirichlet Allokation:
 - ↳ basiert auf Bayes
 - ↳ Wortgruppen suchen, die in verschiedenen Dokumenten häufig vorkommen
 - ↳ diese repräsentieren die Themen
 - ↳ Unterteilung der Bag-Of-Words Matrix in Dokument / Themen und Wort / Themen Matrix
 - ↳ Multiplikation zweier Matrizen
 - ↳ Dokumentenhäufigkeit auf 10% beschränken mit CountVectorizer

Vorhersage stetiger Zielvariablen

- stetige Variablen vorhersagen
- für industrielle, ingenieurtechnische und wissenschaftliche Anwendungen
 - ↳ Verständnis von Zusammenhängen
 - ↳ Erkennen von Trends
 - ↳ Treffen von Vorhersagen

Lineare Regression

- Modellierung zwischen einem oder mehreren Merkmalen mit stetigen Zielvariablen
- überwachtes Lernen
- Beziehung zwischen Regressor x / erklärende Variable und dem Regressanden y / stetige Variable: $y = w_0 + w_1 x$
- Gerade finden, die zur besten Punktwolke passt
- Residuen = vertikaler Abstand zwischen Punkt und Gerade



- Multiple lineare Regression:
 - ↳ mehrere erklärende Variablen
 - ↳ $y = \sum_{m=0}^M w_i x_i = w^T x$

Korrelationsmatrix zur Erkennung von Zusammenhängen

- Quantifizierung des linearen Zusammenhang zwischen den Merkmalen
- quadratische Matrix der Korrelationskoeffizienten $r(x_i, x_j)$
 - ↳ $r=1$ vollständig positiver linearer Zusammenhang zwischen den zwei Merkmalen
 - ↳ $r=-1$ vollständig negativer linearer Zusammenhang
 - ↳ $r=0$ überhaupt kein Zusammenhang

$$r(x_i, x_j) = \frac{\sigma_{x_i x_j}}{\sigma_{x_i} \sigma_{x_j}} = \frac{\sum_{k=1}^n [(x_i^{(k)} - \mu_{x_i})(x_j^{(k)} - \mu_{x_j})]}{\sqrt{\sum_{k=1}^n (x_i^{(k)} - \mu_{x_i})^2} \sqrt{\sum_{k=1}^n (x_j^{(k)} - \mu_{x_j})^2}}$$

- Methode der Kleinsten Quadrate durchführen
- scikit learn: LinearRegression

Behandlung von Ausreißern: robuste Regressionsmodelle

- statistische Tests
- RANSAC - Algorithmus:
 1. Zufällige Stichprobe von "Inliner" wählen
 - ↳ Berechnung der Modellparameter
 2. alle anderen Datenpunkte nehmen und die Abweichungen berechnen (Residuen)
 - ↳ Datenpunkte mit Abweichung unterhalb der Schwelle werden den Inlinern hinzugefügt
 3. neue Berechnung
 4. Fehler Modell mit zu neu abschätzen
 5. Beendigung bei erreichtem Schwellenwert oder Anzahl der Iterationen erreicht
 - ↳ sonst bei 1.
- Hyperparameter: Anzahl der Samples in der Stichprobe, Anzahl der Versuche, Schwellenwerte für Residuen, Art der Abstandsbestimmung
- scikit Learn: RANSACRegressor

Bewertung der Leistung von linearen Regressionsmodellen

- Aufteilung Trainings- und Testdaten
- Residuen-Diagramme:
 - ↳ Funktion der vorhergesagten Werte

- ↳ helfen Ausreißer und Nichtlinearität zu entdecken
- Bestimmtheitsmaß (R^2)
 - ↳ standardisierte Version der mittleren quadratischen Abweichung
 - ↳ SSE: $SSE = \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$
 - ↳ SST: $SST = \sum_{i=1}^n (y^{(i)} - \bar{y})^2$
 - ↳ Bestimmungsmaß: $R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{MSE}{Var(y)}$

Regression und Regularisierung

- Verhinderung von Überanpassung
- Ridge-Verfahren:
 - ↳ neue Straffunktion: $J(w)_{\text{ridge}} = \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2 + \lambda \|w\|_2^2$
 - ↳ Achsenabschnitt wird nicht regularisiert
- LASSO:
 - ↳ neue Straffunktion: $J(w)_{\text{Lasso}} = \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2 + \lambda \|w\|_1$
- ElasticNet Verfahren:
 - ↳ Kompromiss zwischen Ridge und LASSO:
 - ↳ $J(w)_{\text{ElasticNet}} = \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2 + \lambda_1 \sum_{j=1}^m w_j^2 + \lambda_2 \sum_{j=1}^m |w_j|$

Polynomiale Regression

- Umwandeln einer linearen Regression in eine Kurve
- Polynom mit Grad d: $y = w_0 + w_1 x + \dots + w_d x^d$
- Skript: Polynomial Features

Random-Forest-Regression bei nichtlinearen Beziehungen

- polynomiale Regression
- Transformation der Merkmale in einen linearen Zusammenhang
- Entscheidungsbaum-Regression:
 - ↳ keine Transformation der Merkmale
 - ↳ neues Maß für die Unreinheit
 - ↳ muss für stetige Werte geeignet sein
 - ↳ mittlere quadratische Abweichung:

$$I(t) = MSE(t) = \frac{1}{N_t} \sum_{i \in O_t} (y^{(i)} - \hat{y}_t)^2$$

↳ Mittelwert der Stichprobe:

$$\hat{y}_t = \frac{1}{N} \sum_{i \in O_t} y^{(i)}$$

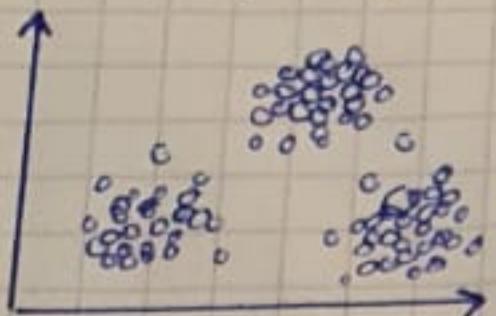
↳ Random Forest Regression gegen Überanpassung

- Idealfall:
 - ↳ Fehler sollten zufällig und unvorhersagbar sein, wenn doch, sind noch Informationen im Diagramm
 - ↳ Verbesserung: Variable-Transformationen, Abstimmungen der Hyperparameter des Lernalgorithmus, Wahl eines einfacheren o. komplexeren Verfahren...

Nicht gekennzeichnete Daten - Unüberwachtes Lernen

- Aufspüren verborgener Strukturen
- Aufspüren von Zentren von Clustern ähnlicher Objekte mithilfe des k-Means-Algorithmus
 - ↳ partitonierender Ansatz
 - ↳ z.B. Kundengruppen mit ähnlichen Interessen
- Prototypbasiertes Clustering:
 - ↳ Cluster werden durch Prototyp repräsentiert
 - ↳ Zentroid: Durchschnitt ähnlicher Punkte mit stetigen Merkmalen

- Medoiden: repräsentativster oder häufigster Punkt
- Vorteile k-Means:
 - ⊕ gut für Identifizierung sphärischer Cluster
 - ⊕ für höherdimensionale Daten
- Nachteile k-Means:
 - ⊖ Anzahl der k Cluster muss festgelegt werden
 - ⊖ ungeeignete k kann Performance herabsetzen



- Algorithmus k-Means:
 1. aus Objekten zufällig k Zentroide als anfängliche Clusterzentren auswählen
 2. alle Objekte dem jeweils nächstem Zentroiden zuweisen
 3. Zentroide neu berechnen
 4. 2. und 3. so oft wiederholen, bis Zuordnung sich nicht mehr ändert, Schwellenwert oder Iterationen erreicht ist

- Ähnlichkeit von Objekten = Gegenteil von Distanz
 - ↪ Distanz = quadrierte euklidische Distanz zweier Punkte x und y im m-dimensionalen Raum
 - ↪ $d(x, y)^2 = \sum_{j=1}^m (x_j - y_j)^2 = \|x - y\|_2^2$
- iterative Ansatz zur Minimierung der Summe der quadratischen Abweichungen innerhalb des Clusters

$$SSE = \sum_{i=1}^k \sum_{j=1}^{n_i} w_{(i,j)} \|x^{(i)} - \mu^{(i)}\|^2$$

- Skalierung bei Daten in verschiedene Größenordnungen oder Min-Max-Skalierung

- eine oder mehrere Cluster können leer sein

- k-Means-++:

↪ anfängliche Zentroide weit voneinander entfernt wählen

↪ bessere Ergebnisse als reines k-Means

↪ Algorithmus:

1. Initialisierung einer leeren Menge M zum Speichern der k auszuwählenden Zentroide
2. aus den Eingabeobjekten einen zufälligen Zentroiden $\mu^{(1)}$ auswählen und in M einfügen
3. alle nicht zu M gehörigen Objekte $x^{(i)}$ ermitteln und die minimale quadrierte Distanz $d(x^{(i)}, M)^2$ zu den Zentroiden aus M konstruieren
4. gewichtete Wahrscheinlichkeitsverteilung: $\frac{d(x^{(i)}, M)^2}{\sum_j d(x^{(i)}, M)^2}$, um den nächsten Zentroiden $\mu^{(P)}$ auszuwählen
5. 2. und 3. wiederholen, bis k Zentroide ausgewählt sind
6. klassischer k-Means-Algorithmus

Harte und weiche Clustering Algorithmen

- hart: jedes Objekt wird genau einem Cluster zugeordnet
- weich: Objekt wird einem oder mehreren Clustern zugewandt
 - ↪ Fuzzy-Clustering-Mean-Algorithmus.
 - 1. Zentroiden festlegen und Punkte einer zufälligen Clusterzugehörigkeit zuordnen

2. Ermittlung Cluster - zentroid
3. Aktualisierung Clusterzugehörigkeit aller Punkte
4. 2. und 3. wiederholen bis Zugehörigkeitskoeffizient sich nicht mehr ändert, Schwellenwert oder Iterationen erreicht ist
 ↳ Fuzzy Koeffizient m : je kleiner dieser wird, umso geringer der Zugehörigkeitsgrad $w_{i,j}$

Optimierung der Anzahl der Cluster k - Ellobogenkriterium

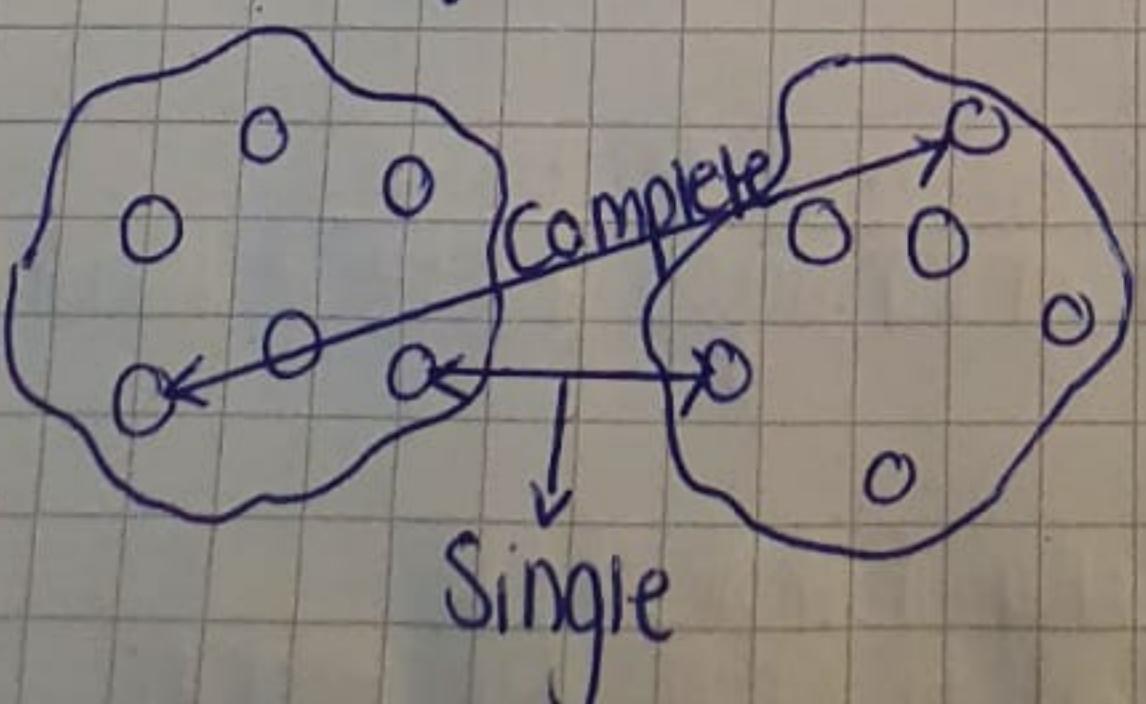
- woher weiß man wie viele Cluster es gibt?
 - inertia - Attribut
 - bei vielen k 's = kleine Werte der Verzerrung (Ellobogen)
 - mit steigenden k sinkt die Verzerrung, da die Objekte sich näher an Zentroide orientieren
 - Optimum = Ellobogen
 - Silhouetten-Diagramm:
 ↳ quantitatives internes Bewertungskriterium für die Clusterin-Güte
1. Berechnung der Geschlossenheit des Clusters $a^{(i)}$
 2. Berechnung der Distanz $b^{(i)}$ zum nächsten Cluster
 3. Berechnung der Silhouette:
$$x^{(i)} = \frac{b^{(i)} - a^{(i)}}{\max(d^{(i)}, b^{(i)})}$$

Cluster als hierarchischen Baum organisieren

- Cluster müssen im Voraus nicht bekannt sein
- Dendrogramme erstellen: Visualisierung binär hierarchischer Cluster
- divisives Clustering:
 - ↳ 1 Cluster umfasst alle Objekte
 - ↳ schrittweises Aufteilen in kleine Cluster
 - ↳ bis jedes Cluster nur noch 1 Objekt enthält
- agglomeratives Clustering:
 - ↳ jedes Objekt ist 1 Cluster
 - ↳ Zusammenführen von nächstgelegenen Clusterpaaren in 1 Cluster
 - ↳ bis 1 Cluster vorhanden ist

Gruppierung von Clustern

- Single Linkage:
 - ↳ Berechnung der Distanz der ähnlichsten Mitglieder für jedes Clusterpaar
 - ↳ beide Cluster, bei denen die Distanz der ähnlichsten Mitglieder am kleinsten ist, werden zusammengeführt
- Complete Linkage:
 - ↳ wie Single, bloß die unähnlichsten Mitglieder



- ↳ Berechnung der Distanzmatrix aller Objekte
- ↳ repräsentieren alle Datenpunkte als Cluster
- ↳ unähnlichste Cluster zusammenführen
- ↳ Aktualisierung Distanzmatrix
- ↳ 3. und 4. wiederholen, bis 1 Cluster bleibt

Aktivierungsfunktionen mehrschichtiger neuronaler Netze

- Problem der Sigmoid Funktion:
 - ↳ große negative Werte x ist Sigmoid fast 0
 - ↳ Netz lernt nur langsam
- Lösung: Tangens hyperbolicus
- Softmax - Funktion:
 - ↳ Abschätzung von Wahrscheinlichkeiten bei Mehrfach-Klassifizierung:

$$P(y=i|z) = \phi(z) = \frac{e^{z_i}}{\sum_{j=0}^M e^{z_j}}$$

- Tangens hyperbolicus:

$$\begin{aligned}\phi_{\text{tan}}(z) &= 2 \cdot \phi_{\text{logistisch}}(z) - 1 \\ &= \frac{e^z - e^{-z}}{e^z + e^{-z}}\end{aligned}$$

- ↳ Aktivierungsfunktion
- ⊕ Back-Propagation wird verbessert

ReLU (Rectified Linear Unit)

- Problem, dass Gradienten verschwinden
- Aktivierungsfunktion: $\phi_{\text{ReLU}}(z) = \max(0, z)$
 - ↳ Negative z : $\phi = 0$
 - ↳ positive z : $\phi = z$

Backward propagation

- man geht von rechts nach links
- Matrix · Vektor = Vektor
↳ schneller als Matrix · Matrix
- 1. Fehler in der Ausgabeschicht bestimmen: $\delta^{out} = a^{out} - y$
- 2. Fehlerterm der verdeckten Schicht: $\delta^n = \delta^{out} (W^{(out)})^T \odot \frac{\partial \phi(z^{(n)})}{\partial z^{(n)}}$
- 3. Formulierung der Straffunktion
- 4. partielle Ableitung der Aktivierungseinheit
- 5. Regularisierung

Training neuronaler Netze mit TensorFlow

- Training von tiefen neuronalen Netzen braucht teilweise sehr lange
 - ↳ Performanceproblem
 - ↳ Verwendung von paralleler Hardware
- Anzahl der Gewichte steigt bei mehreren Schichten
- Training effizienter machen durch die Parallelität der Hardware
- TensorFlow Berechnungen beruhen auf gerichtete Graphen
- GPU mit CUDA
- Low-Level-API:
 - ↳ jeder Knoten repräsentiert eine Operation
 - ↳ Tensor: Abstraktion von Daten; Werte, die entlang der Kanten fließen
 - ↳ Skalar: Tensor, Rang 0
 - ↳ Vektor: Tensor, Rang 1
 - ↳ Matrix: Tensor, Rang 2
 - ↳ "aufgespaltete" Matrizen: Tensor, Rang 3
- 1. Berechnungsgraph erstellen
- 2. Sitzung erstellen und ausführen
 - ↳ Daten in Berechnungsgraph hineinladen
- Bsp.: mit Skalen
 1. Nettoeingabe berechnen: $z = w \cdot x + b \rightarrow$ Bias
 2. Erzeugung des Graphen
 - ↳ Platzhalter für Eingabe-Daten definieren (x)
 - ↳ fixe Eingabe-Daten (w, b)

Array-Strukturen in TensorFlow

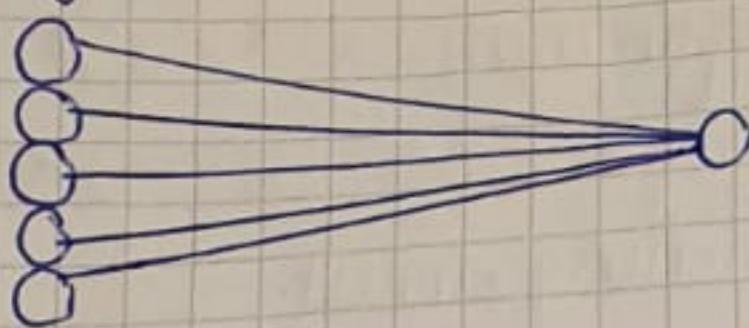
- Bsp. Tensor mit Rang 3: $x[][][]$
- 1. Graph erzeugen mit 2×3 Matrizen
- 2. Operationen: $x=2$, Summe über alle Ebenen $xsum$, Mittelwert über alle Ebenen $xmean$
- 3. Session erzeugen

High Level API von TensorFlow

- Layer API:
 1. Daten einlesen, normalisieren
 2. Modell entwickeln mit Platzhaltern für x, y , Aktivierungsfunktion, Straffunktion
 3. Batchverarbeitung vorbereiten
 4. Training starten
 5. Bewertung des Modells
 - ↳ für schnelle Überprüfung
- Keras API:
 - ↳ für mehrschichtige neuronale Netze

Grundlagen Deep Learning

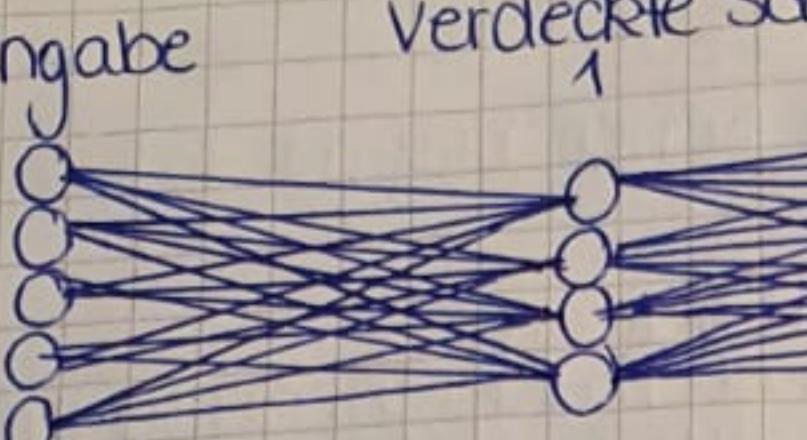
Eingabe



Ausgabe

- Adaline
- einschichtiges Lernen

Eingabe Verdeckte Schicht 1



Verdeckte Schicht 2

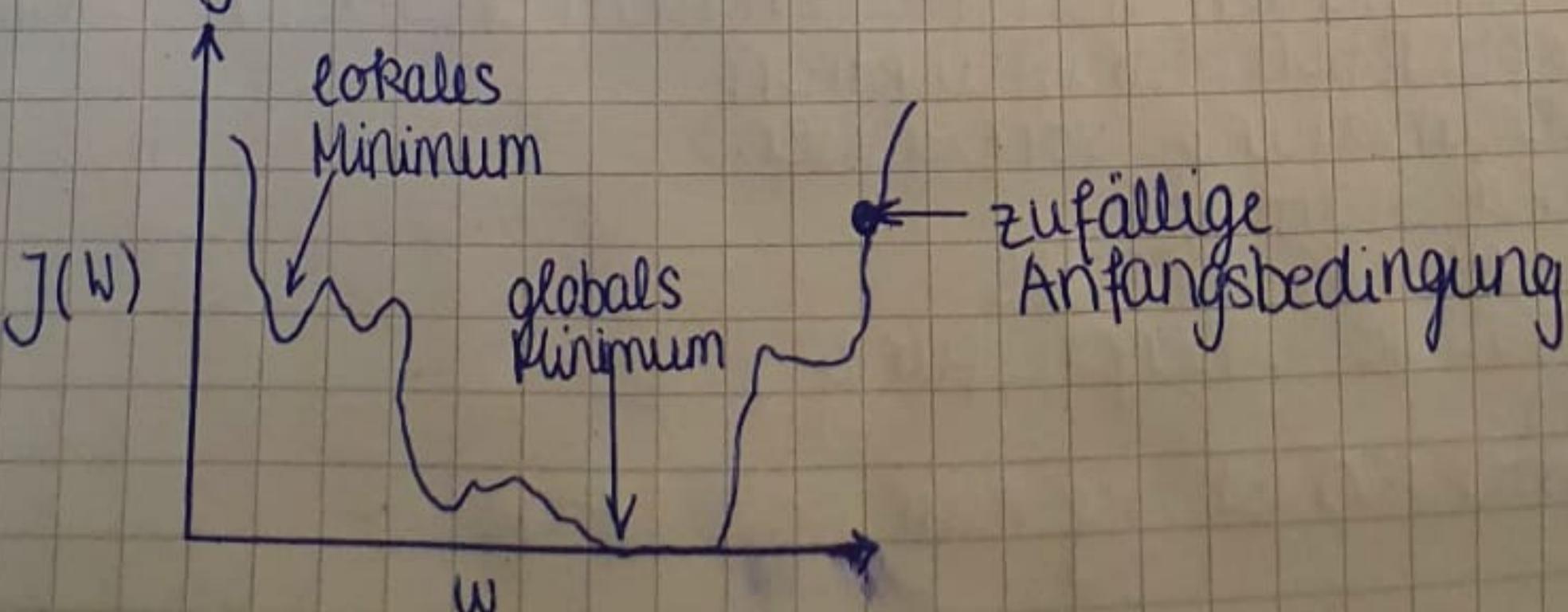
Ausgabe

- mehrschichtiges Netz
- Matrizenrechnung
- Hyperparameter:
 - ↳ Anzahl der Schichten
 - ↳ je mehr Schichten, desto schwerer das Lernen, da Fehlergradienten verkleinert sind

Vorwärtspropagation

1. Eingabeschicht propagiert Trainingsdaten zur Ausgabe
 2. Unhinter der Ausgabe des Netzes lassen sich Fehler berechnen, die man durch Straffunktion minimieren möchte
 3. Fehler wird durch das Netz propagiert und Modell wird aktualisiert
 4. Verfahren wird über mehrere Epochen durchgeführt
- in jeder verdeckten Schicht müssen die Aktivierungselementen bestimmt werden
 - ↳ Nettoeingabe $z_1^{(n)} = a_0^{(in)} w_{0,1}^{(in)} + a_1^{(in)} w_{1,1}^{(in)} + \dots + a_m^{(in)} w_{m,1}^{(in)}$
 - ↳ Aktivierungsfunktion $\phi: a_1^{(n)} = \phi(z_1^{(n)})$
 - ↳ soll differenzierbar sein
 - Sigmoid Funktion verwenden
 - Straffunktion mit und ohne Regularisierung
 - Schicht für Schicht Nettoeingabe und Aktivierungsfunktion mit Matrixenmultiplikation berechnen
 - Minimierungsfunktion der Straffunktion mit Gradientenabstiegsverfahren:

$$\frac{\partial}{\partial w^{(i),j}} J(w)$$
 - mit jeder Schicht gibt es eine neue Matrix
 - Algorithmen dürfen nicht im lokalen Minimum stecken



Clustering durch Bereiche mit hoher Objektdichte

- DBSCAN - Algorithmus
 - Dichte = Anzahl der Objekte innerhalb einer Umgebung mit dem Radius ϵ
 - Verfahren:
 1. Punkte werden speziellen Bezeichnungen zugeordnet
 - ↪ Punkt ist Kernobjekt, Randobjekt oder Rauschobjekt
 - ↪ Kernobjekt: wenn sich innerhalb ϵ min. eine vorgegebene Anzahl weitere Objekte befinden
 - ↪ Randobjekt: wenn sich innerhalb ϵ weniger als MinObj Punkte befinden, der Punkt selbst aber innerhalb von ϵ eines anderen Kernobjekts
 - ↪ Rauschobjekt: alle übrigen Punkte
 2. DBSCAN - Algorithmus
 - ↪ für jedes Kernobjekt oder Gruppe Cluster bilden
 - ↪ Randobjekte dem Cluster zugehörigen Kernobjekte zuweisen
- ⊕ erkennt Habmond-Formen, K-Means nicht
 ⊖ Fluch der Dimensionalität

Einführung Deep Learning - Neuronale Netze

- besteht aus vielen Schichten
- 1940: erste Untersuchungen
- 1986: Backpropagation Verfahren
- z. B. Google Bildersuche, Translate, Facebook Gesichtserkennung, ...

MNIST Datensammlung für Bildverarbeitung

- hat 60000 Objekte in Trainings- und 10000 in Testdatenmenge mit Bildern und Bezeichnungen
- Datenformate:
 - ↪ 28 × 28 Pixel mit Grauwerten
 - ↪ 2-Dimensionale-Daten → 1-Dimension durch Vektor

1. Entpackung der Daten
2. Hilfsfunktion zum Einlesen der Daten und Bezeichner
3. Daten einlesen
4. Visualisierung der Daten
5. Daten zwischenspeichern *

Logistische Regression

- einschichtiges neuronales Netz
- * 6. Logistische Regression

Mehrschichtige neuronale Netzwerkarchitekturen

- einzelne Neuronen in mehrschichtiges Netz zusammenfassen
 - ↪ Feedforward-Netze
- Keras: Python Deep Learning Bibliothek
- verbesserter Ansatz zur logistischen Regression
 - ↪ High-Level-Programmierschnittstelle

Konvolutionale Neuronale Netzwerke

- Faltung
- Funktionsweise der Schrindle
- CNN Architektur
- Bausteine von CNN:
 - ↳ Extraktion auffälliger Merkmale ist für die Leistungsfähigkeit des Lernalgorithmus entscheidend
 - ↳ 1. Extraktion der Low-Level-Merkmale (z.B. Pixel)
 - 2. Low-Level-Merkmale werden schichtweise zu High-Level-Merkmale kombiniert (z.B. Bild)
 - ↳ will gezielte Verknüpfungen erstellen (nur wenige Pixel werden verknüpft)
 - ↳ gemeinsame Parameter mit denselben Gewichtungen
 - ↳ + Reduzierung der Anzahl der Parameter
 - + Einfachere Erfassung auffälliger Parameter
- CNN Aufbau:
 - ↳ mehrere Konvolutionsschichten und Pooling-Schichten (ohne Gewichte und Bias Einheiten)
 - ↳ gefolgt von vollständig verknüpften Schichten

Diskrete Faltung

- Mathematische Notation der Faltung:
 - ↳ grundlegende Operation in einem CNN
- eindimensionale Faltung: (kontinuierlich)
 - ↳ 2 eindimensionale Vektoren $(f * g)(x) = \int_{-\infty}^{\infty} f(x-t)g(t)dt$
 - ↳ Funktion soll $\neq 0$ sein
 - ↳ keine Überlappung der Signale: $|T| > 2 \cdot T_0$ (Produkt ist null)
 - ↳ teilweise Überlappung der Signale: $\Delta T = T_0 - |T|$
 - ↳ $-2T_0 < t < 0$
 - ↳ $0 < t < 2T_0$
- diskrete Faltung:
$$y[i] = \sum_{k=-\infty}^{\infty} x[i-k] w[k]$$

- ↳ eigentlich haben wir endliche Merkmale
 - ↳ unendlicher Vektor durch Padding erstellen durch 0en füllen
 - ↳ negative Indizes
 - ↳ Beispiel eindimensionale Faltung:

$$w = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \quad m=3 \quad (\text{Kernelfunktion}) / \text{Filter}$$

$$x = \begin{bmatrix} 2 & 2 & 2 & 2 \end{bmatrix} \quad n=4 \quad (\text{Eingabevektor})$$

$$x_{pad} = \boxed{0\ 0\ 0\ 2\ 2\ 3\ 2\ 0\ 0\ 0} \quad p=3 \quad i=n+2p \rightarrow i=4+2\cdot 3 = 10$$

$$\begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline \end{array} \quad \begin{aligned} y &= 1 \cdot 0 + 1 \cdot 0 + 1 \cdot 0 = 0 \\ y &= 2 \cdot 1 + 2 \cdot 1 + 2 \cdot 1 = 6 \end{aligned}$$

↳ Beispiel:

$$\begin{array}{r} 32171254 \\ \times \qquad \qquad \qquad \\ \hline \end{array} \qquad * \qquad \begin{array}{r} 1314 \\ \times \qquad \qquad \qquad \\ \hline \end{array}$$

↪ w Filter umdrehen: $\begin{array}{|c|c|c|c|} \hline 4 & 1 & \frac{3}{4} & \frac{1}{2} \\ \hline \end{array}$

↪ Skalarprodukt berechnen: $y_1 = 3 \cdot \frac{1}{4} + 2 \cdot 1 + 1 \cdot \frac{3}{4} + 7 \cdot \frac{1}{2} = 7$

↳ Schrittweite berechnen: $y_2 = 1 \cdot \frac{4}{4} + 7 \cdot 1 + 1 \cdot \frac{3}{4} + 2 \cdot \frac{1}{2} = 9$
↳ 2 hinten

$$\sqrt{3} = 1 \cdot \frac{1}{4} + 2 \cdot 1 + 5 \cdot \frac{3}{4} + 4 \cdot \frac{1}{2} = 8$$

$$y = \boxed{7918} *$$

↪ Kreuzkorrelation: wie oben, aber ohne Umkehrung

- ↪ Kreuzkorrelation: wie oben, aber ohne Umkehrung
- ↪ p wird verwendet, damit alle Elemente berücksichtigt werden

- Full - Padding:

→ erhöhte Anzahl an Dimensionen

→ selten eingesetzt

$$p = m - 1$$

- Same - Padding

→ P hängt von der Größe des Filters ab

→ dm Gebrauchlisten

$$y = x$$

- Valid - Padding:

P-O

→ Kein Padding p=0
Full-Padding oft für Signalverarbeitung, da Randeffekte minimiert werden

- Effekte hinzuaddieren werden
Ausgabegröße berechnen:

$$o = \left[\frac{n+2p-m}{s} \right]_+ + 1 = \left[\frac{8+2 \cdot 4 - 4}{2} \right]_+ + 1 = 7$$

- zweidimensionale diskrete Faltung:

$$Y = X * W \rightarrow Y[i,j] = \sum_{k_1=0}^{\infty} \sum_{k_2=-\infty}^{\infty} X[i-k_1, j-k_2] W[k_1, k_2]$$

↳ Beispiel:

| | | | |
|-----|-----|-----|-----|
| 212 | 0,5 | 0,7 | 0,4 |
| 501 | * | 0,3 | 0,4 |
| 173 | 0,5 | 1 | 0,5 |

1. w. drehen zu

| | | |
|-----|-----|-----|
| 0,5 | 1 | 0,5 |
| 0,1 | 0,4 | 0,3 |
| 0,4 | 0,7 | 0,7 |

S = 2

↳ sehr ineffizient was Speicherbedarf und Komplexität betrifft

Pooling

- Pooling Schicht $P_{n1} \times P_{n2}$ (Nachbarschaft)
- Max-Pooling: max. Werte der Nachbarschaft
- Mean-Pooling: Mittelwert der Nachbarschaft

| 2 1 7 5 0 3 1 7 8 | 1 2 5 4 1 2 3 3 3 | Max | Mean |
|-------------------------|-------------------------|---------------------------------------|---------------------|
| 0 3 2 6 2 5 3 6 0 | 0 1 1 3 0 3 2 1 0 | 8 5 6 3 | 3,78 2,33 3 1,22 |
| | | → lokale Invarianz = weniger Rauschen | |

- ⊕ Anzahl der Merkmale verringern
 - ↳ effizientere Berechnung
 - ↳ Reduzierung der Überanpassung
- eigentlich keine Überlappung

Eigenschaften von Bildern verwenden

- Bilder sind 2 dimensionale Arrays und oft farbig
- 3 Farbkanäle (rot, grün, blau)
- keine Matrizen, sondern Tensoren

$$X_{N_1 \times N_2 \times C_{in}} \quad \left\{ \begin{array}{l} RGB \Rightarrow C_{in}=3 \\ \text{Schwarz - Weiß} \Rightarrow C_{in}=1 \end{array} \right.$$

- Berechnung der Merkmalskarte: $Y^{\text{Faltung}} = \sum_{c=1}^{C_{in}} W[c, :, :] * X[:, :, c]$
- 4 Dimensionen: Breite, Höhe, Cin, Cout

Reduktion der Parameter - Anzahl durch Faltung

- vollständige verknüpfte Schicht ohne Faltung
- Konvolutions-Schicht

Regularisierungsverfahren Drop-out

- für tiefe neuronale Netzwerken
- auf die verdeckten, höheren Schichten angewendet
- während der Trainingsphase wird bei jeder Iteration ein zu fälliger Anteil verdeckter Einheiten entfernt
 - ↳ Wahrscheinlichkeit wird von Benutzer gewählt
- Überanpassung verhindern

Recurrente neuronale Netze

- für sequentielle Daten, z.B. Zeitreihen
- sind bestimmter Reihenfolge angeordnet (Abhängigkeit)
- Repräsentation von Sequenzen:
 - ↳ Sequenzen = Eingabedaten, die abhängig voneinander sind
 - ↳ Darstellung: $(x^{(1)}, x^{(2)}, \dots, x^{(T)})$ Position der Sequenz
 - ↳ Einlernen soll erschaffen werden
- Anwendung:
 - ↳ Sprachübersetzung
 - ↳ automatisches Betteln von Bildern
 - ↳ Texterzeugung

- 1-zu-n: keine Sequenz (Eingabe), Sequenz (Ausgabe)
- n-zu-n: Sequenz (Eingabe und Ausgabe)
 - ↳ synchron: Ein- und Ausgabe gleichzeitig
 - ↳ verzögert: Ein- und Ausgabe verzögert

Sequenzmodellierung mit RNNs

- nur 1 verdeckte Schicht

Ausgabe

verdeckte Schicht $\xrightarrow{\text{rekurrente Kante}}$ (geht auch mehrschichtig)

Eingabe

- Bestimmung der Gewichte
- Berechnung der Aktivierung: $z_h^{(t)} = W_{xh} x^{(t)} + W_{hh} h^{(t-1)} + b_h$
 - ↳ $d_h \cdot \sigma$
- Ausgabe-Einheiten: $h^{(t)} = \phi_h (W_{hy} h^{(t)} + b_y)$
- TBPT:
 - ↳ Gradient wird oberhalb eines Grenzwertes abgeschnitten
 - ↳ explosionsartiges Anwachsen verhindern
- LSTM:
 - ↳ erfolgreicher
 - ↳ Problem des verschwindenden Gradienten lösen
 - ↳ Speicherzelle mit rekurrenter Kante $w=1$

Implementierung mehrschichtiges RNN mit TensorFlow

- Stimmungslage mit tiefen RNNs:
 - ↳ aus Sot+z oder Textdokument geäußerter Meinung
 - ⊕ oder ⊖ beurteilen
 - ↳ Anwendung: Chatbots
 - ↳ 1. Daten einlesen
 - 2. Transformation in nummatische Werte
 - 3. Länge der Bewertungssequenzen behandeln
 - ↳ muss gleiche Länge haben (0 auffüllen oder abschneiden)
 - 4. Unterteilung Trainings- und Testdaten
 - ↳ Einbettung
 - ↳ jedem Wort wird ein reiwertiger Vektor fester Größe zugeordnet
 - ↳ ⊕ Dimensionsreduzierung
 - ⊕ Extraktion auffälliger Merkmale kann trainiert werden
- Sprachmodellierung mit RNNs und TensorFlow
 - ↳ englische Sätze erzeugen
 - ↳ Lernen aus einem Textdokument durch ein Modell
 - ↳ Eingabe: Sequenz von Zeichen
 - ↳ Verarbeitung neuer Zeichen: Berücksichtigung vorangegangener Zeichen
 - ↳ versucht nächstes Zeichen vorherzusagen
 - ↳ 1. Datenaufbereitung
 - 2. Zeichen in Integer umwandeln und zurückverschiebung der Eingabe x und Ausgabe y um ein Zeichen gegenüberliegender \Rightarrow Vorhersage
 - 3. Trainingssatz x in Batches aufteilen
 - 4. Sample erzeugt nächstes Zeichen