

Data Mining

1. Grundlagen

1.1. KDD-Prozess

- KDD- Knowledge Discovery in Databases:

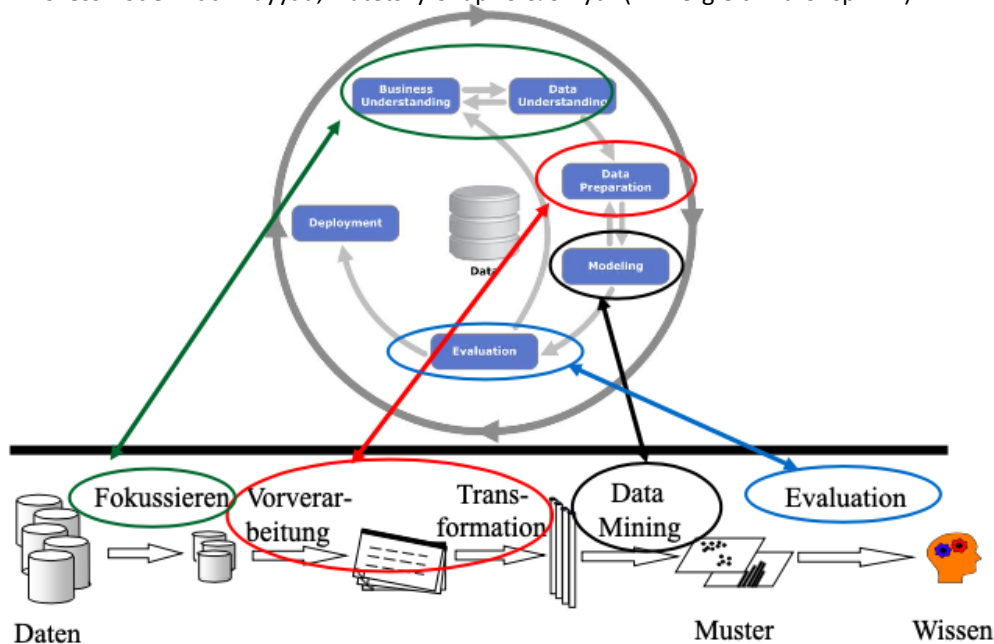
- Prozess der (semi-) automatischen Extraktion von Wissen aus Datenbanken, das gültig, bisher unbekannt, potenziell nützlich und verständlich ist

1.1.1. Das Crisp-DM Modell: Cross Industry Standard Process for Data Mining

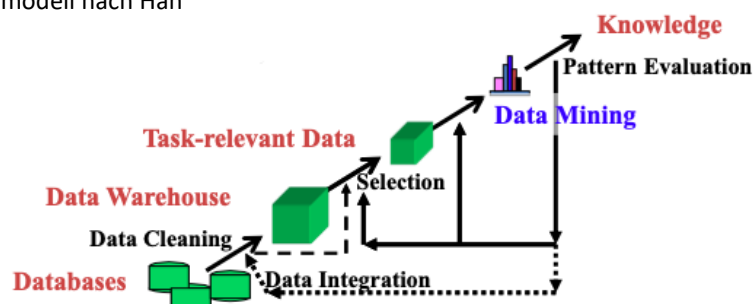
Business, Data Understanding	<ul style="list-style-type: none"> - Verständnis der Anwendungen - Definition der Ziele des KDD - Beschaffung und Selektion der relevanten Daten
Data Preparation	<ul style="list-style-type: none"> - Integration von Daten aus unterschiedlichen Quellen - Konsistenzprüfung, Vervollständigung - Diskretisierung numerischer Attribute - Erzeugen abgeleiteter Attribute
Modeling (Data Mining)	<ul style="list-style-type: none"> - Methoden: Clustering, Klassifikation, Subgruppenentdeckung, Assoziationsregeln - Andere Aufgaben: Regression, Generalisierung, Entdecken von Ausreißern
Evaluation	<ul style="list-style-type: none"> - Präsentation, Bewertung (durch den Benutzer) der gefundenen Muster
Deployment	<ul style="list-style-type: none"> - Planung des Einsatzes, des Monitorings, der Wartung der KDD-Anwendung - Erstellung des endgültigen Berichtes - Review des Produktes

- Alternative Prozessmodelle:

- Prozessmodell nach Fayyad, Piatetsky-Shapiro & Smyth (im Vergleich zu Crisp-DM)



- Prozessmodell nach Han



1.2. Statistik

- Merkmalstypen (Skalenniveau)

Numerisch	Totale Ordnung <, arithmetische Operationen
Ordinal	Mit (totaler) Ordnung <, keine arithmetischen Operationen
Kategorisch/ nominal	Keine Ordnung, keine arithmetischen Operationen

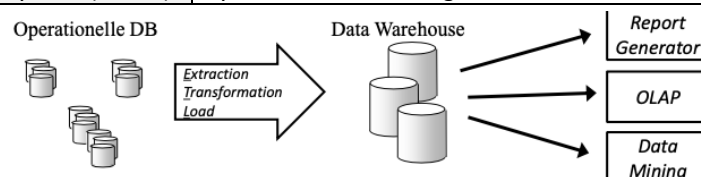
(arithmetischer) Mittelwert	$\bar{x} = \frac{1}{n} * \sum_{i=1}^n x_i$
Median	$x_{med} = \begin{cases} \frac{x_{(n+1)}}{2}, & \text{falls } n \text{ ungerade} \\ \frac{x_{\frac{n}{2}} + x_{\frac{n}{2}+1}}{2}, & \text{falls } n \text{ gerade} \end{cases}$
Mode	Zahl die am häufigsten vorkommt
Varianz	$\sigma^2 = E(x^2) - (E(x))^2$ bzw. $\sigma^2 = \frac{1}{n} * \sum_{i=1}^n (x_i - \bar{x})^2$
Standardabweichung	$\sigma = \sqrt{\sigma^2}$
Kovarianz	$Cov(X, Y) = E((X - E(X)) * (Y - E(Y)))$
Erwartungswert	$E(x) = \sum_i x_i * p(X = x_i)$
Bedingte Wahrscheinlichkeit	$P(A B) = \frac{P(A \cap B)}{P(B)}$
Satz von Bayes	$P(A_i B) = \frac{P(B A_i) * P(A_i)}{P(B)}$
Binomialverteilung „Ziehen mit Zurücklegen“	$P(X = x_j) = \binom{n}{j} * p^j * p^{n-j}$ $E(X) = np$ $V(X) = npq$
Korrelationskoeffizient	$r(X, Y) = \frac{Cov(X, Y)}{\sqrt{V(X) * V(Y)}}$ $ r(X, Y) < 0,5$ schwache Korrelation $0,5 \leq r(X, Y) < 0,8$ mittlere Korrelation $0,8 \leq r(X, Y) $ starke Korrelation

1.3. Datenbanken und Datawarehouse

- Datenbanksystem

Datenbanksystem (DBS)	Softwaresystem zur dauerhaften Speicherung und zum effizienten Suchen in großen Datenmengen
Datenbank (DB)	Sammlung aller Daten und der zugehörigen Beschreibungen
Datenbankmanagementsystem (DBMS)	System zur Verwaltung der Datenbank

- Data Warehouse



- Dauerhafte und integrierte Sammlung von Daten, separiert vom operativen Geschäft aus unterschiedlichen Quellen zum Zweck der Analyse bzw. Entscheidungsunterstützung

1.4. OLAP

OLTP- Online Transaction Processing	OLAP- Online Analytical Processing
Verwendet operative DB	Verwendet Data Warehouse
<ul style="list-style-type: none"> - Hohe Zahl kurzer, atomarer, isolierter, wiederkehrender Transaktionen - Transaktionen benötigen detaillierte, aktuelle Daten (werden häufig aktualisiert) - Transaktionen dienen dem Tagesgeschäft, hoher Anspruch bzgl. Bearbeitungsgeschwindigkeit 	<ul style="list-style-type: none"> - System dient: Der Entscheidungsunterstützung Kann in den Phasen Data Understanding & Preparation eingesetzt werden - Funktionen erlauben: Schnellen, interaktiven Zugriff auf Daten unter „beliebigen“ unternehmensrelevanten Blickwinkeln (Dimensionen) auf verschiedenen Aggregationsstufen

- **OLAP Multidimensionalität: (Data Cube)**
 - o Multidimensionales Modell für ein Data Warehouse, mit flexiblen, interaktiven Aggregations-bzw. Verfeinerungsfunktionen entlang einer/ mehrere Dimensionen(en)
 - o Unterscheidet zwei Typen von Attributen: Dimensionen (unabhängige Att)(z.B. Zeit) und Maße (abhängige Att) , Attribute können hierarchisch angeordnet sein

- **OLAP Operationen:**

Drill-Down/ Roll-Up Operationen	Visualisierung verschiedener Aggregationsstufen Drill-Down: Übergang zur nächsthöheren Generalisierungsebene Roll-Up: Übergang zur nächstniedrigeren Generalisierungsebene
Slice & Dice- Operationen	Setze Bedingungen für dargestellte Daten →Reduziere die Dimensionalität der visualisierten Daten

- Bekannte mehrdimensionale Datenmodelle: Sternschema, Schneeflockenschema

1.5. Vorverarbeitung - Data Preprocessing:

- Auswahl der Daten: Erstellen einer Datentabelle + Reduktion der Datenmenge
- Data Cleaning: Datenkonsistenz, fehlerhafte, fehlende Werte, Duplikate, ...
- Anpassung der Daten an die Data Mining Verfahren:
Diskretisierung, Normalisierung, Feature Selection*, Feature Construction**
- **Single Table Assumption**
 - o **Propositionalisierung:** Transformation einer relationalen Datenbank in eine einzige Tabelle
 - o Nutzt Aggregationen

- | |
|--|
| <ul style="list-style-type: none"> - *Feature Selection <ul style="list-style-type: none"> o Unterscheidung: Relevantes, Irrelevantes, Redundantes Feature o Filtere: <ul style="list-style-type: none"> ▪ Anachronismen: Attribute die zum Vorhersagezeitpunkt unbekannt sind ▪ monoton steigende Features (Zeit, ID,...) ▪ Features mit sehr wenigen non-default Werten und sehr vielen unterschiedlichen Werten o Suche eine ideale Teilmenge der Featuremenge bzgl. einer Bewertungsfunktion (2ⁿ Teilmengen!) - **Feature Construction <ul style="list-style-type: none"> o Konstruiere aus gegebenen Features neue Werte o Sehr große Anzahl an Möglichkeiten für neue Features, oft wird Hintergrundwissen genutzt |
|--|

- **Instance Selection**
 - o Entfernen fehlerhafter Datensätze (Alternative: korrigieren)
 - o **Sampling:**

Random Sampling (Normalfall)	(Zufälliges) Auswählen einer Teilmenge der Daten
Stratified Sampling	Erhöhe Anteil der Instanzen aus der seltenen Klasse (vor allem bei „Imbalanced Data“)

- Korrelationen zwischen Attributen sollten sich auch in den zufälligen Teilmengen wiederfinden

- **Diskretisierung**

Equal-Width Discretization:	Alle Intervalle sind gleich groß
Equal-Frequency Discretization:	Alle Intervalle enthalten gleich viele Instanzen
Supervised Discretization:	Beziehe ein anderes (z.B. binäres) Attribut mit in die Diskretisierung ein

- o Ein Intervall soll möglichst (im binären Fall) „nur positive“ oder „nur negative“ Beispiele enthalten

Top-down	Starte mit einem Intervall, Teile sukzessive in Teile auf die möglichst zur gleichen Klasse gehören
Bottom-up	Starte mit jedem Wert auf, die möglichst zur gleichen Klasse gehören

- **Normalisierung:**

- o Vereinheitlichung der Wertebereiche (oft auf das Intervall [0;1])

Lineare min-max Normalisierung	$v' = \frac{v - \min A}{\max A - \min A}$
z-score Normalisierung	$v' = \frac{v - \text{mean} A}{\text{sd} A}$

2. Clustering

2.1. Grundlagen des Clusterings

- Ziel: Daten (semi-) automatisch so in Kategorien, Klassen oder Gruppen (Cluster) einteilen, dass Objekte im gleichen Cluster möglichst ähnlich und Objekte aus verschiedenen Clustern möglichst unähnlich zueinander sind

2.1.1. Distanz- und Ähnlichkeitsfunktionen

- Distanz ist indirekt proportional zu Ähnlichkeit
- Adäquatheit der Distanzfunktion bestimmt die Qualität des Clusters, Wahl hängt von Anwendung ab
- **Distanzfunktion:** Funktion: $\text{dist}(\cdot, \cdot): O \times O \rightarrow [0, \infty)$ mit folgenden Eigenschaften:
 1. $d(o_1, o_2) = d \in \mathbb{R}^{\geq 0}$
 2. $d(o_1, o_2) = 0$ genau dann, wenn $o_1 = o_2$
 3. $d(o_1, o_2) = d(o_2, o_1)$ (Symmetrie)
 - gilt zusätzlich: 4. $d(o_1, o_3) \leq d(o_1, o_2) + d(o_2, o_3)$, dann heißt dist : metrische Distanzfunktion

- Distanzfunktion für numerische Attribute

- Gegeben: 2 Objekte der Form: $x = (x_1, \dots, x_d)$ im d-dimensionalen Raum

Allgemeine L_p – Metrik (Minkowski – Distanz)	$\text{dist}(x, y) = \sqrt[p]{\sum_{i=1}^d x_i - y_i ^p}$
Euklidische Distanz ($p=2$)	$\text{dist}(x, y) = \sqrt{\sum_{i=1}^d x_i - y_i ^2}$
Manhattan-Distanz ($p=1$)	$\text{dist}(x, y) = \sum_{i=1}^d x_i - y_i $
Maximums- Metrik ($p= \infty$)	$\text{dist}(x, y) = \max \{ x_i - y_i 1 \leq i \leq d\}$

- Populäre Ähnlichkeitsfunktion: Korrelationskoeffizient $\in [-1, +1]$

- Distanzfunktion für kategorische Attribute:

Hamming- Abstand	$\text{dist}(x, z) = \sum_{i=1}^d \delta(x_i, y_i)$, mit $\delta(x_i, y_i) = \begin{cases} 0 & \text{falls } x_i = y_i \\ 1 & \text{sonst} \end{cases}$ „zählt“ Anzahl der unterschiedlichen Attributwerte
Jaccard-Metrik	Abstandsmaß für Mengen von Werten (z.B. boolesche Attribute) $\text{dist}(x, y) = \frac{ x \cup y - x \cap y }{ x \cup y }$ Analog dazu das Ähnlichkeitsmaß Jaccard- Koeffizient. $\text{sim}(x, y) = \frac{ x \cap y }{ x \cup y }$

- Typen von Clusterverfahren:

Partitionierende Verfahren (pV)	<ul style="list-style-type: none"> - Parameter: Distanzfunktion für Punkte, Anzahl k der Cluster - Sucht ein „flaches“ Clustering in k Cluster mit minimalen Kosten (jedes Cluster min. ein Objekt, jedes Objekt in genau einem Cluster)
Dichtebasierte pV	<ul style="list-style-type: none"> - Parameter: Distanzfunktion für Punkte, minimale Dichte in einem Cluster - Erweitert Cluster um Nachbarn, solange die Dichte groß genug ist
Hierarchische Verfahren	<ul style="list-style-type: none"> - Parameter: Distanzfunktion für Punkte und Cluster - Bestimmt Hierarchie von Clustern, nicht nur ein Clustering
Andere Clusterverfahren	Subspace Clustering (Suche Cluster in Teilmengen von Attributen), Konzeptuelle Cluster (Suche Cluster mit verständlichen Beschreibungen), Fuzzy Clustering, Graphen- theoretische Verfahren, Neuronale Netze

2.2. Partitionierende Verfahren

- Lokal optimierendes Verfahren
 - Wähle k initiale Cluster- Repräsentanten
 - Optimierte diese Repräsentanten iterativ
 - Ordne jedes Objekt seinem ähnlichsten Repräsentanten zu

- Typen von Cluster – Repräsentanten:
 - o Mittelwert des Clusters (Konstruktion eines zentralen Punktes)
 - o Element des Clusters (Auswahl eines repräsentativen Punktes)
 - o Wahrscheinlichkeitsverteilung des Clusters (Erwartungsmaximierung)

2.2.1. k-means

- populärste (partitionierende) Clustering- Verfahren

Objekt	Punkt $p = \{x_1^p, \dots, x_d^p\}$ in einem euklidischen Vektorraum
Clustering \mathbb{C}	$\mathbb{C} = \{C_1, \dots, C_k\}$ von k Clustern weist jeden Punkt p einem Cluster C_i zu
Centroid μ_C	Mittelwert aller Punkte im Cluster C $\mu_C = (\bar{x}_1(C), \bar{x}_2(C), \dots, \bar{x}_d(C))$, wobei $\bar{x}_j(C) = \frac{1}{n_C} * \sum_{p \in C} x_j^p$, der Mittelwert der j-ten Dimension aller Punkte in C, n_C = Anzahl der Objekte in C
$TD^2(C)$	Maß für die Kosten (Kompaktheit) eines einzelnen Clusters C: $TD^2(C) = \sum_{p \in C} \text{dist}(p, \mu_C)^2$ Je kleiner TD^2 desto näher liegen die Punkte, umso kompakter ist C
TD^2	Maß für die Kosten (Kompaktheit) eines Clusterings: $TD^2 = \sum_{i=1}^k TD^2(C_i)$ Je kleiner TD^2 für ein Clustering, desto kompakter sind durchschnittlich die einzelnen Cluster des Clusterings

- o Finden eines optimalen Clustering ist NP-schwer

- **Verfahren:**

k-means	Parameter: Punktmenge D, Integer k
1.	Erzeuge eine „initiale“ Zerlegung der Punktmenge D in k Clustern Berechne die Menge $C' = \{C'_1, \dots, C'_k\}$ der Centroide für die k Cluster
2.	Bilde k Cluster durch Zuordnung jedes Punktes zum nächstliegenden Centroid aus C
3.	Berechne die Menge $C' = \{C'_1, \dots, C'_k\}$ der Centroide für die neu bestimmten Cluster Wenn ein Punkt p vom Cluster C_1 zum Cluster C_2 wechselt, werden die Koordinaten der zugehörigen Centroide folgendermaßen inkrementell angepasst: - $\bar{x}(C'_1) = \frac{1}{n_{C_1}-1} * (n_{C_1} * \bar{x}_j(C_1) - x_j^p)$ - $\bar{x}(C'_2) = \frac{1}{n_{C_2}+1} * (n_{C_2} * \bar{x}_j(C_2) + x_j^p)$
4.	Wiederholung ab Schritt 2- bis sich die Lage der Zentren nicht mehr ändert return Centroide

Vorteile	Nachteile
Aufwand: $O(n)$ für eine Iteration, Anzahl der Iterationen ist im Allgemeinen klein (~5-10)	Anfälligkeit für Rauschen + Ausreißern, da alle Objekte in die Berechnung des Centroids eingehen
Einfache Implementierung	Cluster müssen konvexe Form haben
Gute Verständlichkeit	Anzahl k der Cluster ist oft schwer zu bestimmen
Effizienz	Starke Abhängigkeit von der initialen Zerlegung (Ergebnis und Laufzeit) und Reihenfolge

- Varianten:

Algorithmus von Lloyd	Algorithmus wie zuvor angegeben: Clusterzentren werden nach einer vollständigen Iteration neu berechnet
Algorithmus von MacQueen	Wenn ein Punkt einem neuen Cluster zugeordnet wird, so werden die Clusterzentren <u>sofort</u> aktualisiert (→ Algo= Reihenfolgenabhängig)

- o In der Praxis haben beide Varianten ähnliche Eigenschaften

2.2.2. k-medoids /PAM (Partitioning Around Medoids)

- Setzt nur Distanzfunktion für Paare von Objekten voraus (auch möglich für ordinale/ kategorielle Attribute)
- Greedy-Algorithmus

Medoid m_c	zentrales Element des Clusters (\approx Centroid, ist aber immer existierender Datenpunkt Durch eine Menge von Medoiden M ist genau ein Clustering einer Datenmenge D gegeben
$TD(C)$	Maß für die Kosten (Kompaktheit) eines Clusters C : $TD(C) = \sum_{p \in C} \text{dist}(p, m_c)$
TD	Maß für die Kosten (Kompaktheit) eines Clusterings (mit k Clustern): $TD = \sum_{i=1}^k TD(C_i)$
Suchraum	Alle k -elementigen Partitionen der Datenbank D mit $ D =n$

- Laufzeitkomplexität: $O(n^3 \text{ (Initialisierung)} + k * (n - k)^2 \text{ (je Iteration)})$
- Ergebnisse oft besser als k-means aber langsamer k-means

- Verfahren

k-medoids	Parameter: Objektmenge D , Integer k
1.	Initialisiere die k Medoide Berechne TD der Initialisierung
2.	Berechne TD Wert für jede mögliche Vertauschung eines Medoids mit einem anderen Objekt der Datenmenge for each Medoid m do for each Nicht-Medoid n do Berechne Kosten $TD_{m \leftrightarrow n}$, dies ich ergeben, wenn m und n vertauscht werden
3.	Wenn der kleinste TD-Wert besser ist als der TD-Wert des aktuellen Clusterings, dann wird die durch die entsprechende Vertauschung entstehende Menge von Medoiden zum aktuellen Clustering
4.	Wiederhole ab Schritt 2 solange, bis sich der TD-Wert des aktuellen Clusterings nicht mehr Verbessern lässt Return Medoide

2.2.3. **k-modes**: Clustering mit kategorischen Daten:

- Initialisierung: nicht zufällig, sondern k Objekte aus der Datenmenge als initiale Modes
- Distanzfunktion: Distanzfunktion für Datensätze mit kategorischen Attributen (z.B. Hamming-Distanz)
- Cluster- Repräsentanten: Modes
Mode einer Menge von Vektoren $\{X_1, \dots, X_n\}$: Q , dass $\sum_{X_i} \text{dist}(X_i, Q)$ minimiert

2.2.4. **Erwartungsmaximierung**

- Idee: Punkt gehört zu mehreren Clustern mit unterschiedlicher Wsk.
- Objekt - Punkte $p = \{x_1^p, \dots, x_d^p\}$ in einem euklidischen Vektorraum
- Ein Cluster wird durch eine Wsk.verteilung beschrieben (Normal/Gaußverteilung) nicht durch einen repräsentativen Punkt wie zuvor
- Repräsentation eines Clusters C
 - Mittelwert μ_C aller Punkte des Clusters
 - $d \times d$ Kovarianzmatrix Σ_C für die Punkte im Cluster C

$P(x C)$	Wsk.dichte eines Clusters bzw Wsk. mit der bei einer einzigen gegebenen Gaußverteilung C ein Punkt x in der Datenmenge vorkommt	$P(x C) = \frac{1}{\sqrt{(2\pi)^d \Sigma_C }} e^{-\frac{1}{2}(x-\mu_C)^T (\Sigma_C)^{-1} (x-\mu_C)}$
W_i	Größe eines Clusters/ Anteil der Punkte aus dem Datensatz D	$W_i = \frac{1}{n} \sum_{x \in D} P(C_i x)$
$P(C_i x)$	Wsk.dichte eines Clusterings $M = \{C_1, \dots, C_k\}$, Wsk., dass ein Objekt zu einem bestimmten Cluster C_i gehört	$P(C_i x) = W_i * \frac{P(x C_i)}{P(x)}$
$P(x)$	Zuordnung von Punkten zu Clustern:	$P(x) = \sum_{i=1}^k W_i * P(x C_i)$
$E(M)$	Maß für die Güte (Wsk.) eines Clustering M	$E(M) = \sum_{x \in D} \log(P(x))$
	Je größer der Wert E , desto wahrscheinlicher sind die gegebenen Daten D , unter Annahme der berechneten Verteilung, $E(M)$ soll maximiert werden	

- **Verfahren:**

EM	Parameter: Punktmenge D, Integer k
1.	Erzeuge ein „initiales“ Modell $M = (C_1', \dots, C_k')$ (von Gaußverteilungen für D)
2.	Neuzuordnung Berechne $P(x C_i)$, $P(x)$ und $P(C_i x)$ für jedes Objekt aus D und jede Gaußverteilung/ jedes Cluster C_i
3.	Neuberechnung des Modells Berechne ein neues Modell $M' = \{C_1, \dots, C_k\}$ durch Neuberechnung von W_i , μ_C und Σ_C für jedes $i = 1 \dots k$ $W_i = \frac{1}{n} \sum_{\vec{x} \in D} P(C_i \vec{x})$ $\bar{\mu}_i = \frac{\sum_{\vec{x} \in D} \vec{x} * P(C_i \vec{x})}{\sum_{\vec{x} \in D} P(C_i \vec{x})}$ $\Sigma_i = \frac{\sum_{\vec{x} \in D} P(C_i \vec{x}) (\vec{x} - \bar{\mu}_i)(\vec{x} - \bar{\mu}_i)^T}{\sum_{\vec{x} \in D} P(C_i \vec{x})}$
4.	Wiederhole ab Schritt 2, bis sich die Güte der beiden Modelle M und M' nicht mehr stark unterscheidet (höchstens um einen benutzerdefinierten Wert ϵ) $ E(M) - E(M') < \epsilon$ return M

- Diskussion

- Konvertiert gegen ein (möglicherweise nur lokales) Minimum
- Laufzeitkomplexität je Iteration: $O(n * |M|)$ (Anzahl der benötigten Iterationen sehr hoch)
- Ergebnis und Laufzeit hängen stark ab von der initialen Zuordnung und von der Wahl von k
- Modifikation für Partitionierung der Daten in k disjunkte Cluster: jedes Objekt wird dem Cluster zugeordnet, zu dem es am wahrscheinlichsten gehört

- **Wahl der initialen Cluster:**

- Idee: Clustering einer kleinen Stichprobe liefert im Allgemeinen gute initiale Cluster
 - Problem: Einzelne Stichproben sind evtl. deutlich anders verteilt als die Grundgesamtheit
- Methode (Fayyad, Reina & Bradley)
 - Ziehe unabhängig voneinander m verschiedene Stichproben
 - Clustere jede der Stichproben: \rightarrow m verschiedene Schätzungen für k Clusterzentren:
 $A = (A_1, \dots, A_k), B = (B_1, \dots, B_k), C = (C_1, \dots, C_k) \dots$
 - Clustere nun die Menge $DB = A \cup B \cup C \cup \dots$ mit m verschiedenen Initialisierungen A, B, C, ...
 - Wähle von den m Clusterings das mit dem besten Wert bezüglich eines Gütemaßes

- **Wahl des Parameters k**

- Idee: Bestimme für $k = 2, \dots, n-1$ jeweils ein Clustering und wähle das „beste“ aus
- Benötigt: **Maß für die Güte eines Clusterings:**

- Gütemaß- Davies- Bouldin Index (DB)

Güte innerhalb des Clusters C_i	$S_i = \sqrt[q]{\frac{1}{ C_i } \sum_{x \in C_i} \text{dist}(x, \mu_i)^q}$
Güte der Trennung der Cluster C_i und C_j	$M_{i,j} = \text{dist}(\mu_i, \mu_j)$
$R_{i,j}$ für $i \neq j$	$R_{i,j} = \frac{S_i + S_j}{M_{i,j}}$
Davis-Bouldin Index	$DB = \frac{1}{k} \sum_{i=1}^k D_i \quad \text{mit } D_i = \max_{i \neq j} R_{i,j}$

- Gütemaß- Silhouetten- Koeffizient

- Von k unabhängiges Gütemaß für k-means und k-medoid
- Sei $a(o)$ Abstand eines Objektes o zum Repräsentanten seines Clusters und
 $b(o)$ Abstand zum Repräsentanten des nächstgelegenen Clusters (exklusive eigenes), dann:

$$\text{Silhouette } s(o) \text{ des Objekts } o: s(o) = \begin{cases} 0 & \text{wenn } |C| = 1, \text{ d. h. } a(o) = 0 \\ \frac{b(o) - a(o)}{\max\{a(o), b(o)\}} & \text{sonst (Wertebereich } [-1; 1]) \end{cases}$$

$s(o) < 0$	o liegt näher am Repräsentanten eines anderen Clusters \rightarrow schlechte Zuordnung
$s(o) \approx 0$	o liegt zwischen zwei Clustern \rightarrow indifferente Zuordnung
$s(o) \text{ nahe } 1$	o liegt nahe am eigenen Clusterrepräsentanten \rightarrow gute Zuordnung

- Silhouetten- Koeffizient s_C eines Clusters C = durchschnittliche Silhouette aller Objekte

2.2.5. Mögliche Problemfälle:

- | | |
|----|--|
| 1. | Cluster sind nicht kugelförmig, sondern haben stark unterschiedliche Ausdehnungen in verschiedenen Richtungen des Raumes |
| 2. | Cluster haben stark unterschiedliche Größen |
| 3. | Cluster haben stark unterschiedliche Punktdichte |
- Mit k-means und k-medoid kann kein Rauschen erkannt werden, da implizit der gesamte Datenraum durch die repräsentativen Punkte aufgeteilt wird.
 - EM ist etwas robuster bzgl. Punktdichte und Größe der Cluster

2.3. Dichte-basiertes Clustern

- Idee: Cluster als Gebiete im d-dimensionalen Raum, in denen die Objekte dicht beieinander liegen getrennt, durch Gebiete, in denen die Objekte weniger dicht liegen
- Anforderungen an dichtebasierte Cluster:
 - Für jedes Objekt eines Clusters überschreitet die lokale Punktdichte in einen gegebenen Grenzwert
 - Die Menge von Objekten, die das Cluster ausmacht, ist räumlich zusammenhängend

Kernobjekt	Objekt $o \in O$, für das gilt: $ N_\epsilon(o) \geq \text{MinPts}$, wobei $N_\epsilon(o) = \{o' \in O \mid \text{dist}(o, o') \leq \epsilon\}$ → wenn in ϵ Umgebung von o mindestens MinPts viele Objekte liegen Objekte, die keine Kernobjekte sind, können zu einem Cluster gehören (Randpunkte), oder auch nicht (Rauschen).
direkt dichte-erreichbar	Objekt $p \in O$ ist direkt dichte erreichbar von $q \in O$ bzgl. ϵ und MinPts, wenn gilt: $p \in N_\epsilon(q)$ und q ist ein Kernobjekt in O Alle Objekte, die in der ϵ Umgebung eines Kernobjekts p liegen, sind direkt dichte erreichbar von p
dichte-erreichbar	Objekt p ist dichte-erreichbar von q , wenn es eine Kette von direkt dichte-erreichbaren Objekten zwischen q und p gibt. Alle Objekte in dieser Kette (außer evtl. p) sind dabei Kernobjekte
dichte-verbunden	Zwei Objekte p und q sind dichte-verbunden, wenn sie beide von einem dritten Objekt o aus dichte-erreichbar sind Das verbindende Objekt o muss nicht in jedem Fall verschieden von p oder q sein (zwei direkt dichte- erreichbare Objekte p und q sind dichte-verbunden (etwa über q))
Cluster C bzgl. ϵ und MinPts	=nicht-leere Teilmenge von O , für die die folgende Bedingungen erfüllt sind: Maximalität: $\forall p, q \in O$: wenn $p \in C \wedge q$ dichte erreichbar von p , dann ist auch $q \in C$ Verbundenheit: $\forall p, q \in C$: p ist dichte – verbunden mit q
	<u>Dichte-basiertes Clustering CL</u> der Menge O bzgl. ϵ und MinPts ist Menge aller dichte-basierten Cluster bzgl. ϵ und MinPts in O
Noise _{CL} (Rauschen)	Menge aller Objekte aus O , die nicht zu einem der dichte-basierten Cluster C_i gehören
Grundlegende Eigenschaft	Sei C ein dichte-basiertes Cluster und sei $p \in C$ ein Kernobjekt, dann gilt $C = \{o \in O \mid o \text{ dichte – erreichbar von } p \text{ bzgl. } \epsilon \text{ und MinPts} \}$

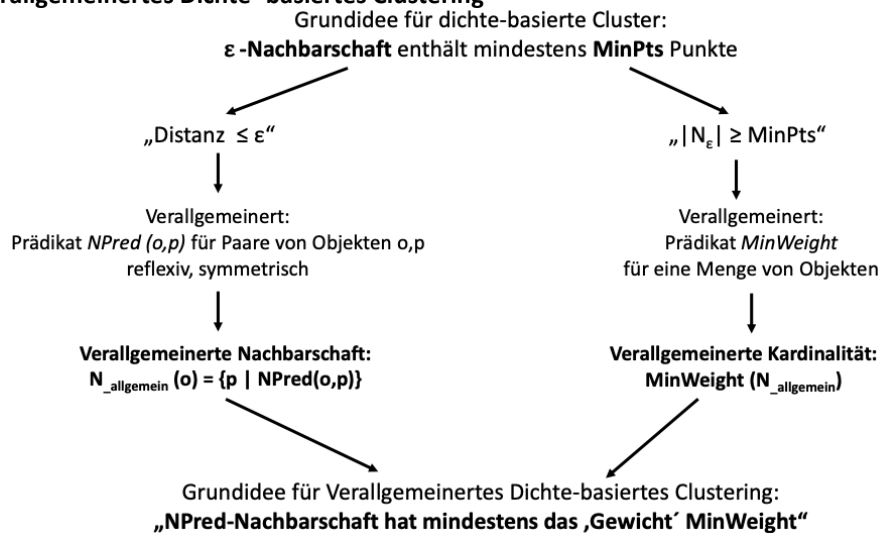
DBSCAN	Parameter SetOfPoints, ϵ , MinPts
	Zu Beginn: alle Objekte unklassifiziert Datenmenge linear durchgehen, ausgehend von jedem unklassifizierten Objekts versuchen einen kompletten Cluster zu finden (Mit Expandiere Cluster (EC))
EC	Parameter SetOfPoints, Point, ClId, ϵ , MinPts
1.	Überprüfen ob das aktuelle Objekt ein Kernobjekt ist Falls aktuelles Objekt != Kernobjekt, wird es dem Rauschen zugeordnet und return false
2.	Falls das aktuelle Objekt ein Kernobjekt ist, kann wegen der oben stehenden grundlegenden Eigenschaft ein neuer Cluster gefunden werden, indem alle von ihm aus dichte-erreichbaren Objekte in der Datenbank gesucht werden
3.	Alle Objekte in der ϵ Umgebung des aktuellen Objekts gehören auf jeden Fall zum Cluster, da sie direkt dichte-erreichbar sind
4.	Die von diesen Objekten wiederum direkt dichte-erreichbaren Objekte gehören ebenfalls auch zum Cluster, da sie transitiv dichte-erreichbar sind
5.	Es wird also durch iterierte Berechnung der direkten Dichte-Erreichbarkeit der gesamte Cluster zum aktuellen Kernpunkt gefunden. Alle diese Objekte werden mit der gleichen ClusterId ClId markiert

- Parameterbestimmung

- Cluster: Dichte größer als die durch ϵ und MinPts spezifizierte „Grenzdichte“
- Gesucht: Der am wenigsten dichte Cluster in der Datenmenge
- Heuristische Methode: betrachte Distanzen zum k-nächsten Nachbarn
- Funktion k-Distanz: Distanz eines Objekts zu seinem k-nächsten Nachbarn
- k-Distanz-Diagramm: Die k-Distanzen aller Objekte absteigend sortiert
- Probleme:
 - Hierarchische Cluster
 - Stark unterschiedliche Dichte in verschiedenen Bereichen des Raums
 - Cluster und Rauschen sind nicht gut getrennt

MinPts:	<ul style="list-style-type: none"> - MinPts= 1 oder MinPts=2 können zum Single-Link-Effekt führen - Heuristik, die sich experimentell in vielen Anwendungen bewährt hat, ist , den Wert für MinPts gleich $(2*d)$ zu setzen, mit d= Dimension des Datenraums
ϵ	Für ein gegebenes k wäre ein guter Wert für ϵ die k- Distanz eines Objekts im „dünnsten“ Cluster mit einer für diesen Cluster hohen k-Distanz (geringste Dichte, bei der man noch einen Cluster annimmt und nicht schon Rauschen)

- Verallgemeinertes Dichte- basiertes Clustering



2.4. Hierarchisches Clustering

- Ziel: Konstruktion einer Hierarchie von Clustern, sodass immer die Cluster mit minimaler Distanz verschmolzen werden
- Im Gegensatz zu partitionierenden Verfahren wird keine einfache Zerlegung der Datenmengen erzeugt, sondern eine hierarchische Repräsentation der Daten

2.3.1. Dendrogramm

- Baum, der die hierarchische Zerlegung der Datenmenge O in immer kleinere Teilmengen darstellt
- Knoten repräsentieren jeweils ein Cluster, Kante hat als Attribut die Distanz zwischen den Objekten
- Wurzel repräsentiert die ganze Datenbasis, Blätter die einzelnen Objekte (Datenpunkte)
- Skala, die die Distanz zwischen den Clustern angibt

2.3.2. Algorithmen: Hierarchisches Clustern

- 2 Typen von hierarchischen Verfahren: Bottom-Up (agglomerative) und Top-Down (divisive)

- Algorithmus: Agglomeratives Hierarchisches Clustern (Aufwand $O(n^2)$)

1.	Bilde initiale Cluster, die jeweils aus einem Objekt bestehen, und bestimme die Distanzen zwischen allen Paaren dieser Cluster.
2.	Bilde einen neuen Cluster aus den zwei Clustern, welche die geringste Distanz zueinander haben
3.	Bestimme die Distanz zwischen dem neuen Cluster und allen anderen Clustern
4.	Wenn alle Objekte sich in einem einzigen Cluster befinden: Fertig, andernfalls wiederhole ab Schritt 2

- Distanzfunktionen für Cluster

- Gegeben: Distanzfunktionen $\text{dist}(x,y)$ zwischen zwei Datenpunkten x und y
- Typische Distanzfunktionen:

Single Link	$\text{dist} - \text{sl}(X, Y) = \min_{x \in X, y \in Y} \text{dist}(x, y)$
Complete Link	$\text{dist} - \text{cl}(X, Y) = \max_{x \in X, y \in Y} \text{dist}(x, y)$
Average Link	$\text{dist} - \text{al}(X, Y) = \frac{1}{ X \cdot Y } * \sum_{x \in X, y \in Y} \text{dist}(x, y)$

Vorteile	Nachteile
Erfordert keine Kenntnis der Anzahl k der Cluster	Entscheidungen können nicht zurückgenommen werden
Findet nicht nur ein flaches Clustering, sondern eine ganze Hierarchie	Anfällig gegenüber Rauschen (Single-Link) z.B. eine „Linie“ von Objekten kann zwei Cluster verbinden
Ein einzelnes Clustering kann aus dem Dendrogramm D gewonnen werden, z.B. mit Hilfe eines horizontalen Schnitts durch D	Ineffizient: Laufzeitkomplexität von mindestens $O(n^2)$ für n Objekte

3.3.3. Dichte-basiertes Hierarchisches Clustern

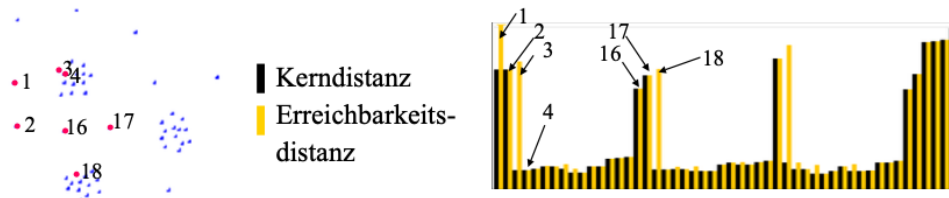
- Grundidee: in einem DBSCAN-ähnlichen Durchlauf gleichzeitig das Clustering für verschiedene Dichte-Parameter bestimmen

Kerndistanz eines Objekts o bzgl. ϵ und MinPts	$\text{Kerndistanz}_{\epsilon, \text{MinPts}}(o) = \begin{cases} \text{undefiniert,} & \text{wenn } N_{\epsilon}(o) < \text{MinPts} \\ \text{Distanz}_{\text{MinPts}}(o), & \text{sonst} \end{cases}$ also die kleinste Distanz, bei der das Objekt o ein Kernobjekt ist. d.h. mindestens MinPts Punkte liegen in der Umgebung mit Radius $\text{Kerndistanz}(o)$ um o und in jedem kleineren Radius liegen noch nicht MinPts viele Punkte.
$\text{Distanz}_k(o)$	Distanz zum k -nächsten Nachbarn von o
Erreichbarkeitsdistanz eines Objekts p relativ zu einem Objekt o	$\text{Erreichbarkeitsdistanz}_{\epsilon, \text{MinPts}}(p, o) = \begin{cases} \text{undefiniert, wenn } N_{\epsilon}(o) < \text{MinPts} \\ \max\{\text{Kerndistanz}(o), \text{dist}(o, p)\}, & \text{sonst} \end{cases}$ Kleinste Distanz bei der p von o aus direkt dichte- erreichbar ist, sofern diese kleiner als ϵ ist. Mindestens so groß wie die Kerndistanz von o

- OPTICS

OPTICS	OPTICS-update
Eingabe: Datenpunkte D , Maximale Distanz ϵ , minimale Anzahl Elemente für einen Kernpunkt MinPts	Eingabe: Prioritätenliste Q , Kernpunkt p , Maximale Distanz ϵ , minimale Anzahl Elemente für einen Kernpunkt MinPts
Ausgabe: Sortierte Liste L	
Algorithmus $Q = \text{Leere Prioritätenliste}^1$ Für alle p in D Falls $p.\text{verarbeitet} == \text{FALSCH}$ $p.\text{kerndistanz} = \text{Kerndistanz}_{\epsilon, \text{MinPts}}(p)$ $p.\text{erreichbarkeitsdistanz} = \text{UNDEFINIERT}$ $p.\text{verarbeitet} = \text{WAHR}$ Füge p in L ein Falls $p.\text{kerndistanz} \neq \text{UNDEFINIERT}$ $\text{update}(Q, p, \epsilon, \text{MinPts})$ Solange $Q \neq []$ $o = Q.\text{nächster}()$ $o.\text{kerndistanz} = \text{Kerndistanz}_{\epsilon, \text{MinPts}}(o)$ $o.\text{verarbeitet} = \text{WAHR}$ Füge o in L ein Falls $o.\text{kerndistanz} \neq \text{UNDEFINIERT}$ $\text{update}(Q, o, \epsilon, \text{MinPts})$	Algorithmus $\text{dist}_p = p.\text{kerndistanz}$ $N_p = \text{Nachbarn}_{\epsilon}(p)$ Für alle o in N Falls $o.\text{verarbeitet} == \text{FALSCH}$ $\text{dist}_o = \text{Erreichbarkeitsdistanz}_{\epsilon, \text{MinPts}}(o, p)$ Falls $o.\text{erreichbarkeitsdistanz} == \text{UNDEFINIERT}$ $o.\text{erreichbarkeitsdistanz} = \text{dist}_o$ Füge o in Q ein Sonst, falls $\text{dist}_o < o.\text{erreichbarkeitsdistanz}$ $o.\text{erreichbarkeitsdistanz} = \text{dist}_o$ Aktualisiere Priorität von o in Q

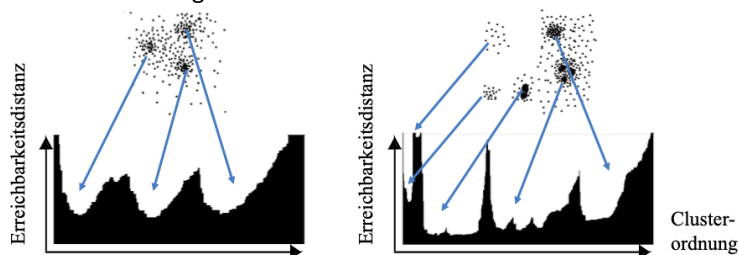
- liefert kein hierarchisches Clustering, sondern eine Clusterordnung (Reihenfolge der Objekte)
- Clusterordnung bzgl. ϵ und MinPts
 - Clusterordnung = beginnt mit einem beliebigen Objekt
 - Als nächstes wird das Objekt besucht, das zur Menge der bisher besuchten Objekte die minimale Erreichbarkeitsdistanz besitzt



- Dichtere Gebiete sind durch kürzere Balken zu erkennen
- Größere Sprünge in der Clusterordnung von einem Cluster zum nächsten erkennt man daran, dass der erste Punkt des zweiten Clusters, da er schon zu einem dichteren Gebiet gehört, eine kleine Kerndistanz, aber, da er von einem entfernt liegenden Punkt in der Clusterordnung erreicht wurde, eine hohe Erreichbarkeitsdistanz besitzt (z.B. Punkte 3 und 18)

- Erreichbarkeitsdiagramm

- Durchlaufe alle Objekte in der durch die Clusterordnung gegebenen Reihenfolge
- Für jedes Objekt: Zeige die Erreichbarkeitsdistanz (ϵ und MinPts) der Objekte als senkrechte, nebeneinander liegende Balken



- Parametersensitivität
 - Clusterordnung ist robust gegenüber den Parameterwerten
 - Gute Resultate, wenn Parameterwerte „groß genug“

3.4. Clustering hochdimensionaler Daten

- Hochdimensionale Daten = sehr viele Attribute für Datenpunkte
- Problem: **Curse of Dimensionality**
 - Daten mit vielen Dimensionen schwierig zu interpretieren und visualisieren
 - Attribute können korreliert sein → Distanzfunktionen werden „verbogen“
 - Hinzufügen von Dimensionen „streckt Daten“ auseinander
 - Relativer Abstand zwischen maximalen und minimalen Abstand konvergiert für steigende Anzahl an Dimensionen $\lim_{d \rightarrow \infty} \frac{\text{dist}_{\max} - \text{dist}_{\min}}{\text{dist}_{\min}} \rightarrow 0$
 - Cluster können in höherdimensionalen Räumen überlappen

2.4.1. Subspace Clustering

- Aufgabe: Finde gute Cluster in beliebigen Teilräumen
- Problem: Es gibt exponentiell viele (Achsen-parallele) Teilräume; Jede Teilmenge an Attributen induziert einen Teilraum
- Strategien:

Top down	Starte mit Clustern im Gesamttraum (alle Attribute), projiziere auf Teilräume -> keine Garantie für die besten Cluster
Bottom up	Beginne mit Clustern im 1-Dimensionalen; verbinde Cluster zu höherdimensionalen Clustern -> hohe Komplexität

- CLIQUE Algorithmus (Clustering In QUEst)

- Gitter und dichte basiertes Verfahren für Bottom-up Subspace Clustering
 - Jede Dimension wird in gleich große Intervalle aufgespalten → „1-dimensionale Einheiten“
 - Eine d-dimensionale Einheit ist der Schnitt von d 1-dimensionalen Einheiten
 - Eine d-dimensionale Einheit ist dicht, wenn sie mindestens τ Objekte enthält
 - Ein subspace cluster ist die maximale Menge von verbundenen k-dimensionalen Einheiten
- Nutze: Anti-Monotonie: Wenn d-dimensionale Einheit dicht ist, dann sind alle enthaltenen (d-1)-dimensionalen Einheiten auch dicht (reduziert die Zahl der zu untersuchenden Teilräume)

- Finde alle 1-dimensionalen dichten Einheiten
- In Iteration $d > 1$: generiere alle k-dimensionalen dichten Einheiten
 - Neuer Kandidat: Vereinige zwei (d-1) dichte Einheiten, die sich nur in einer Einheit unterscheiden
 - Entferne Kandidaten („Pruning“): Falls ein Kandidat eine Teilmenge besitzt, die nicht dicht ist, dann ist der Kandidat auch nicht dicht
 - Überprüfe die Dichte des Kandidaten durch Auszählen aus der Datenbank
- Output: Paare (O, D): O ist eine Menge von Objekten, D ist eine d-dimensionale Einheit
- Post-Processing: Verbinde d-dimensionale Einheiten, die nebeneinander liegen

3. Assoziationsregeln

3.1. Einfache Assoziationsregeln

- Regeln der Form: Rumpf \rightarrow Kopf [support, confidence]

Items	Elemente der Menge $I = \{i_1, \dots, i_m\}$ (Attribute, Spalte in der Tabelle) Sind in Transaktionen oder Itemsets lexikographisch sortiert: Itemset $X = (x_1, \dots, x_k)$, wobei $x_1 \leq x_2, \dots \leq x_k$
Itemsets	Teilmenge $X \subseteq I$ (Attributmenge, ein oder mehrere Spalten in der Tabelle)
Transaktion	Menge von Items $T \subseteq I$ (Fälle, Zeilen in der Tabelle)
Datenbank D	Multimenge von Transaktionen $D = \{T_1, \dots, T_n\}$ (Menge von Zeilen)
$X \subseteq T$	„T enthält X“; Die Transaktion T enthält das Itemset X
Länge des Itemsets	Anzahl der Elemente in einem Itemset (k-Itemset: Itemset der Länge k)
Support	Support des Items X in D := Anteil der Transaktionen in D, die X enthalten $\text{supp}_D(X) = \frac{ \{T \in D X \subseteq T\} }{ D }$ Frequency: Absolute Häufigkeit der Menge X: $\text{supp}_D(X) * D = \{T \in D X \subseteq T\} $ Support einer Assoziationsregel: Support der Vereinigung $X \cup Y$ in D
Konfidenz	Konfidenz c einer Assoziationsregel $X \rightarrow Y$ in D: Anteil der Transaktionen, die die Menge Y enthalten in der Teilmenge aller Transaktionen aus D, welche die Menge X enthalten $\text{conf}_D(X \rightarrow Y) = \frac{\text{supp}_D(X \cup Y)}{\text{supp}_D(X)}$

- Schnellere Bestimmung häufiger Itemsets:
 - Bestimmung des Supports durch Zählen auf der Datenbank (Scan)
 - Anti-Monotonie-Eigenschaft: Jede Teilmenge eines häufigen Itemsets ist ebenfalls häufig
 - Vorgehen:

1.	Bestimme erst 1- Itemsets, dann 2-Itemsets, dann 3-Itemsets...
2.	Finden von k+1-elementigen häufigen Itemsets (nur Itemsets betrachten, für die k-elementige Teilmengen häufig auftreten)
3.	Nur für diese Kandidaten bestimme den Support durch Zählen in der Datenbank Kandidaten: Obermenge aller häufig vorkommenden Itemsets der Länge k - Wesentlich kleiner als die Menge aller k-elementigen Teilmengen von I

3.1.1. Apriori-Algorithmus

- C_k : Die zu zählenden Kandidaten- Itemsets der Länge k
- L_k : Menge aller häufig vorkommenden Itemsets der Länge k

Apriori	Parameter: I, D, minsup
1.	Bestimme die einelementigen Itemsets, die minimalen Support haben
2.	Nun werden aus den im jeweils vorhergehenden Durchlauf bestimmten häufig auftretenden Itemsets (mit k-1 Elementen) neue Kandidaten- Itemsets (mit k Elementen) gebildet Der Support dieser Kandidaten wird gezählt, indem für jede Transaktion T in der Datenbank (mithilfe Subset Funktion) geprüft wird, welche der aktuellen Kandidaten in T enthalten sind. Für diese Kandidaten-Itemsets c wird der Zähler c.count, der den Support für c zählt um 1 erhöht
3.	Es werden diejenigen Itemsets für den nächsten Durchlauf ausgewählt, die minimalen Support haben. return $\bigcup_k L_k$

- Bestimmung von Kandidaten der Länge k:

1. Join	- k-1-elementige häufige Itemsets p und q werden miteinander verbunden, wenn sie in den ersten k-2 Items übereinstimmen (1,2,3) (1,2,4) → (1,2,3,4)
2. Pruning (Abschneiden)	- Entferne alle Kandidaten-Itemsets, die eine k-1-elementige Teilmenge von Items enthalten, die nicht zu L_{k-1} gehört Teilmenge von (1,2,3,4) müssen in L_{k-1} vorkommen sonst wird entfernt

- Effizienzverbesserung:

- Verwaltung der Kandidaten in einem speziellen Hashbaum
- Reduktion der Transaktionen:
 - Transaktion, die kein häufiges k-Itemset enthält, wird nicht mehr benötigt → Löschen
- Partitionierung der Datenbank:
 - Ein Itemset ist nur dann häufig, wenn es in mindestens einer Partition häufig ist
 - Bilde Hauptspeicherresidente Partitionen der Datenbank
- Sampling:
 - Anwendung des gesamten Algorithmus auf ein Sample
 - Zählen der gefundenen häufigen Itemsets auf der gesamten Datenbank
 - Feststellen evtl. weiterer Kandidaten und Zählen auf der gesamten Datenbank

3.1.2. **FP-Growth**

- Komprimiert den Datensatz + Vollständige Information zur Bestimmung der Frequent Items
- Ablauf in 2 Phasen:
 1. Aufbau des initialen FP-Trees
 2. Generiere aus dem FP-Tree Conditional Trees und die häufigen Items
- Aufbau FP-Trees

1. Durchlauf	Sortiere Items absteigend nach Häufigkeit (→sorgt für stärkere Kompression des FP-Trees) Items, die nicht den minimalen Support haben, werden entfernt (Für FP-Growth wird der absolute Support genutzt)
2. Durchlauf	Füge Fälle sukzessive in Baum ein: <ul style="list-style-type: none"> - Starte an der Wurzel des Baumes - Suche den Nachfolger-Knoten für das erste noch nicht benutzte Item - Falls Nachfolger existieren, erhöhe count, sonst erstelle neuen Knoten - Alle Knoten zu einem Item werden mit „Auxiliary Links“ verknüpft - Die „Header Tabelle“ zählt für jedes Item die Häufigkeit mit

- **Kandidatengenerierung**

- 1-Itemsets stehen direkt in der Header Table
- Für K-Itemsets: Divide and conquer
 - Suche erst alle Häufigen Itemsets, die mit „C“ enden, dann alle, die mit „BD“ enden...
 - Verwende Bottom- up Algorithmus: Von den Blättern zur Wurzel
 - Baue dazu Conditional Trees:
 - Suche alle Präfix-Pfade zu einer Endung (Auxiliary Links)
 - Füge diese dann in einem neuen Baum ein
 - Der count für jeden Präfix Pfad entspricht dem Count im untersten Knoten
 - Rekursiver Aufruf auf den Conditional Trees

Vorteile	Nachteile
Benötigt nur 2 Durchläufe auf der Datenbank	FP-Tree passt unter Umständen nicht in den Hauptspeicher
Komprimiert den Datensatz	In manchen Fällen sind FP-Trees „teuer“ aufzubauen („dichte“ Daten)
Keine explizite Kandidatengenerierung (oft) schneller als Apriori	(Kompliziertere Implementierung)

- **Bestimmung der Assoziationsregeln**
 - o Sei X ein häufig vorkommendes Itemset
 - o Für jede Teilmenge A von X bestimme die Konfidenz der Regel $A \rightarrow X \setminus A$

$$\text{confidence}(A \rightarrow (X - A)) = \frac{\text{support}(X)}{\text{support}(A)}$$
 - o Streiche die Regel, wenn die Konfidenz kleiner als minConf ist
 - o Speicherung der häufigen Itemsets mit ihrem Support in einer Hashtabelle
 -> keine Datenbankzugriffe nötig

3.2. Auswahl von Assoziationsregeln

- **Interessantheitsmaß:** Je größer der Wert für eine Regel ist, desto interessanter ist der Zusammenhang
- (Beispiel-) Interessantheitsmaß für die Regel $A \rightarrow B$: Added Value

$$\frac{\text{sup}(A \wedge B)}{\text{sup}(A)} - \text{sup}(B) = \text{conf}(A \rightarrow B) - \text{sup}(B)$$

„um wieviel steigt die Wahrscheinlichkeit von B, wenn die Bedingung A hinzugefügt wird?“

- **Vierfeldertafel:**

	A	\bar{A}	Σ
B	P(A, B)	P(\bar{A} , B)	P(B)
\bar{B}	P(A, \bar{B})	P(\bar{A} , \bar{B})	P(\bar{B})
Σ	P(A)	P(\bar{A})	1

- **Kriterien für Interessantheitsmaße:**

Conciseness	Kürzere Regeln sind besser (weniger Items)
Generality	Generellere Regeln sind besser (mehr Fälle abdecken)
Reliability	Hohe confidence/accuracy besser
Diversity	Regeln sollten untereinander unähnlich sein
Novelty	Vorher unbekannt, nicht aus anderen Regeln ableitbar
Unexpectedness	Gute Regeln widersprechen Vorwissen/ Erwartungen
Applicability	Kann praktisch (in der Anwendung) umgesetzt werden

- **Constraints:**

- o Bedingungen, die zusätzlich oder alternativ zu Interessantheitsmaßen verwendet werden können

Domain Constraints	$S\theta, v \theta \in \{=, \neq, <, >, \geq\}$ $v\theta S, \theta \in \{\in, \notin\}$ $v\theta S$ oder $S\theta V, \theta \in \{\subseteq, \subset, \not\subseteq, =, \neq\}$
Aggregations Constraint $\text{agg}(S)\theta v$ mit	$\text{agg} \in \{\min, \max, \text{sum}, \text{count}, \text{avg}\}$ $\theta \in \{=, \neq, <, >, \geq\}$

- **Anti-Monotonie von Constraints**

- o Können verwendet werden, um die Kandidatengenerierung zu beschleunigen
- o Support ist ein anti-monotones Constraint

Monotonie	Anti-Monotonie
Ein Constraint C ist monoton, wenn (und nur wenn) für alle Itemsets S und S' gilt: Falls $S \supseteq S'$ und S das Constraint C verletzt, dann verletzt auch S' das Constraint C	Ein Constraint C ist anti-monoton, wenn (und nur wenn) für alle Itemsets S und S' gilt: Falls $S \supseteq S'$ und S das Constraint C erfüllt, dann erfüllt auch S' das Constraint C

Antimonoton	Monoton	Teilweise
$S\theta v \theta \in \{=, \leq, \geq\}$ $S \subseteq V$ $\min(S) \geq v$ $\max(S) \leq v$ $\text{count}(S) \leq v$ $\text{sum}(S) \leq v$ (frequent constraint)	$v \in S$ $S \supseteq V$ $\min(S) \leq v$ $\max(S) \geq v$ $\text{count}(S) \geq v$ $\text{sum}(S) \geq v$ $\text{avg}S \ v \theta \in \{=, \leq, \geq\}$	$S = V$ $\min(S) = v$ $\max(S) = v$ $\text{count}(S) = v$ $\text{sum}(S) = v$

3.3. Hierarchische Assoziationsregeln

- Vorteil: Items, die höher in der Hierarchie stehen, haben höheren Support (Allgemeinere Aussagen)

Items	$I = \{i_1, \dots, i_m\}$, Menge von Literalen
H	gerichteter, azyklischer Graph über der Menge von Literalen I (Taxonomie) H repräsentiert eine Menge von is-a Hierarchien über den Items in I
Kante in H von i nach j	i heißt Vater oder direkter Vorgänger von j, j heißt Sohn oder direkter Nachfolger von i
Weg in H von i nach k	i ist Vorfahre von k (auch: i ist Generalisierung von k) k ist Nachfahre von i (auch: k ist Spezialisierung von i)
Vorfahre V	Menge von Items V heißt Vorfahre einer Menge von Items N, wenn man V aus N dadurch bilden kann, dass man mindestens ein Item in N durch seinen Vorfahren ersetzt
D	Menge von Transaktionen T, wobei $T \subseteq I$ Typischerweise: Transaktionen T enthalten nur Items aus den Blättern des Graphen H
Transaktion T unterstützt ein Item $i \in I$	i ist in T enthalten oder i ist ein Vorfahre eines Items, das in T enthalten ist
T unterstützt eine Menge $X \subseteq I$ von Items	T unterstützt jedes Item in X Anpassung: Ein Item wird für eine Transaktion gezählt, wenn es entweder selbst enthalten ist, oder wenn mindestens ein Nachfahre der Taxonomie im Baum enthalten ist
Support einer Menge $X \subseteq I$ von Items in D	Prozentsatz der Transaktionen in D, die X unterstützen

- **Definition Hierarchische Assoziationsregeln:**
 - o $X \rightarrow Y$ mit $X \subseteq I, Y \subseteq I, X \cap Y = \emptyset$
Kein Item in Y ist Vorfahre eines Items in X bezüglich H (da $x \rightarrow \text{Vorfahre}(x) = 100\% = \text{redundant}$)
- **Bestimmung der häufig auftretenden Itemsets:**
 - o Grundidee: Erweiterung der Transaktionen um alle Vorfahren von enthaltenen Items
 - o Methode:
 - Jedes Item in einer Transaktion wird zusammen mit all seinen Vorfahren bezüglich H in eine neue Transaktion T' eingefügt
 - Es werden keine Duplikate eingefügt
- **Cumulate Algorithmus**
 - o Basiert auf Apriori
 - o Anpassungen/ Verbesserungen:
 - Vorberechnung von Vorfahren eines Items
 - Filtern der hinzugefügten Vorfahren
 - nur diejenigen Vorfahren zu einer Transaktion hinzufügen die in einem Element der Kandidaten Menge C_k des aktuellen Durchlaufs auftauchen
 - Filtern redundanter Items
 - generiere keine Kandidaten die ein Item und eine Generalisierung des Items beinhalten
 - X hat denselben Support wie $X \cup i$, wenn X einen Nachfahren von i enthält
 $X \cup i$ muss nicht generiert werden
- **Startifikation**
 - o Alternative zum Basisalgorithmus (Apriori)/ Cumulate-Algorithmus
 - o Startifikation: "Schichtenbildung" der Mengen von Itemsets
 - o Anti-Monotonie der Hierarchie: Itemset V hat keinen minimalen Support und N ist Nachfolger von $V \rightarrow N$ hat keinen minimalen Support

- Methode:
 - nicht mehr alle Itemsets einer bestimmten Länge auf einmal zählen, sondern erst die generelleren Itemsets zählen; die spezielleren nur dann, wenn nötig
 - Stratifikation liegt folgende Beziehung zwischen Itemsets zugrunde:
 - Wenn ein Itemset X' keinen minimalen Support hat und X' Vorfahre von X ist, dann hat auch X keinen minimalen Support
- **Interessantheit Hierarchischer Assoziationsregeln:**
 - $X' \rightarrow Y'$ ist Vorfahre von $X \rightarrow Y$
 Das Itemset X' ist Vorfahre des Itemset X UND/ODER
 Das Itemset Y' ist ein Vorfahre der Menge Y
 - $X' \rightarrow Y'$ ist direkter Vorfahre von $X \rightarrow Y$ in einer Menge von Regeln:
 $X' \rightarrow Y'$ ist Vorfahre von $X \rightarrow Y$ und es existiert keine Regel $A \rightarrow B$, sodass $A \rightarrow B$ Vorfahre von $X \rightarrow Y$ und $X' \rightarrow Y'$ ein Vorfahre von $A \rightarrow B$ ist
- Hierarchische Assoziationsregel $X \rightarrow Y$ heißt **R- interessant**, wenn die
 - Keine direkten Vorfahren hat ODER Tatsächlicher Support $> (=)$ dem R-fachen des erwarteten Supports ODER Alternativ: tatsächliche Konfidenz $>$ dem R-fachen der erwarteten Konfidenz
- **Variabler Support:**
 - Problem:
 - Minsup zu hoch \rightarrow keine Low-Level- Assoziation
 - Minsup zu niedrig \rightarrow zu viele High-Level- Assoziationen
 - Idee: Verwende unterschiedliche minimalen Support auf unterschiedlichen Taxonomie- Ebenen
 - Vorteil: Effektivität
 - Nachteil: Effizienz

3.4. Quantitative Assoziationsregeln

- Idee: Datenbank mit numerischen und kategorischen Attributen so zu transformieren, dass ein ähnliches Verfahren wie für boolesche Datenbanken angewendet werden kann
- Lösungsansätze:

Statische Diskretisierung	Dynamische Diskretisierung
<ul style="list-style-type: none"> - Diskretisierung aller Attribute vor dem Bestimmen von Assoziationsregeln (im Pre-Processing) - Ersetzung numerischer Werte durch Bereiche/ Intervalle - Verwende Standard- Verfahren auf diskretisierten Daten 	<ul style="list-style-type: none"> - Diskretisierung der Attribute beim Bestimmen von Assoziationsregeln (z.B. Maximiere Interessantheitsmaß) \rightarrow neues Verfahren - Zusammenfassen „benachbarter“ Assoziationsregeln zu einer verallgemeinerten Regel

Attribute I	$I = \{i_1, \dots, i_m\}$ eine Menge von Literalen
$I_V = I \times \mathbb{N}^+$	Menge von Attribut- Wert-Paaren, d.h. ein Paar $\langle x, v \rangle$ steht für ein Attribut x mit dem zugehörigen ganzzahligen Wert v
D	Menge von Datensätzen $R, R \subseteq I_V$ Jedes Attribut darf höchstens einmal in einem Datensatz vorkommen
I_R	$\{ \langle x, u, o \rangle \in I \times \mathbb{N}^+ \times \mathbb{N}^+ \mid u \leq o \}$ $\langle x, u, o \rangle$ Ein Attribut x mit einem zugehörigen Intervall von Werten $[u \dots o]$ Attribute(X) für $X \subseteq I_R$: die Menge $\{x \mid \langle x, u, o \rangle \in X\}$
Quantitative Items	die Elemente aus I_R
Quantitatives Itemset	Menge $X \subseteq I_R$
Datensatz R unterstützt eine Menge $X \subseteq I_R$	Zu jedem $\langle x, u, o \rangle \in X$ gibt es ein Paar $\langle x, v \rangle \in R$ mit $u \leq v \leq o$
Support der Menge X in D für ein quantitatives Itemset X	Prozentsatz der Datensätze in D , die X unterstützen
Quantitative Assoziationsregel:	$X \rightarrow Y$ mit $X \subseteq I_R, Y \subseteq I_R$ und Attribute(X) \cap Attribute(Y) = \emptyset

Support s einer quantitativen Assoziationsregel	Support der Menge $X \cup Y$ in D
Konfidenz c einer quantitativen Assoziationsregel	Prozentsatz der Datensätze, die die Menge Y unterstützen, in der Teilmenge aller Datensätze, welche auch die Menge X unterstützen
Verallgemeinerung eines Itemsets S	Itemset V (S Spezialisierung von V) 1. V und S enthalten die gleichen Attribute 2. Die Intervalle in den Elementen von S sind vollständig in den entsprechenden Intervallen von V enthalten

- Methode
 - Diskretisierung numerischer Attribute
 - Transformation kategorischer Attribute in binäre numerische Attribute (Werte: 1–0)
 - Bestimmung des Supports für jedes einzelne Attribut-Wert-Paar in D
 - Zusammenfassung „benachbarter Attributwerte“ zu Intervallen, solange der Support der entstehenden Intervalle kleiner ist als $\maxsup \rightarrow$ Häufig vorkommende 1-Itemsets
 - Finden aller häufig auftretenden quantitativen Itemsets: \rightarrow Variante des Apriori-Algorithmus
 - Bestimmen quantitativer Assoziationsregeln aus häufig auftretenden Itemsets
 - Entfernen aller uninteressanten Regeln: $< \min\text{-interest}$ (bzgl. eines gewählten Interessantheitsmaßes)
- **Partitionierung numerischer Attribute**
 - Probleme
 - Minimaler Support: Zu viele Intervalle \rightarrow zu kleiner Support für jedes einzelne Intervall
 - Minimale Konfidenz: Zu wenig Intervalle \rightarrow zu kleine Konfidenz der Regeln
 - Lösung
 - Zerlegung des Wertebereichs in viele Intervalle
 - Zusätzliche Berücksichtigung aller Bereiche, die durch Verschmelzen benachbarter Intervalle entstehen
 - Durchschnittlich $O(n^2)$ viele Bereiche, die einen bestimmten Wert enthalten

4. Klassifikation

4.1. Einleitung

4.1.1. Klassifikationsproblem:

- Gegeben: eine Menge $O \subset D$ von Objekten des Formats (o_1, \dots, o_d) mit Attributen $A_i, 1 \leq i \leq d$ und Klassenzugehörigkeit $c \in C = \{c_1, \dots, c_k\} \rightarrow$ Überwachtes Verfahren
- Gesucht: Die Klassenzugehörigkeit für Objekte aus $D \setminus O$, ein Klassifikator $K: D \rightarrow C$
- Verwandtes Problem: Vorhersage (Prediction)
 - Gesucht ist der Wert für ein numerisches Attribut
- Prozess der Klassifikation: 1. Trainingsphase; 2. Anwendungsphase

4.1.2. Trainieren und Testen:

- **Train- Validation- Test:** Aufteilung der Menge O in zwei Teilmengen (z.B. 50/20/30)
 1. Trainingsmenge zum Lernen des Klassifikators
 2. Validierungsmenge zum Optimieren der Parameter
 3. Testmenge zum Bewerten des Klassifikators
 - Benötigt viele Beispiele mit bekannter Klassenzugehörigkeit
- **k- fold cross- validation** (k- fache Kreuzvalidierung) anwendbar auch für wenige bekannte Beispiele
 - Teile die Menge O in k gleich große Teilmengen
 - Verwende jeweils $k-1$ Teilmengen zum Trainieren, die verbleibende Teilmenge zum Testen
 - Kombiniere die erhaltenen k Klassifikationsfehler (und die gefundenen k Modelle)

4.1.3. Gütemaße für Klassifikation

- Sei K ein Klassifikator, $TR \subseteq O$ die Trainingsmenge, $TE \subseteq O$ die Testmenge. Bezeichne $C(o)$ die tatsächliche Klasse eines Objekts o

Klassifikationsgenauigkeit (classification accuracy) von K auf TE	$G_{TE}(K) = \frac{ \{o \in TE \mid K(o) = C(o)\} }{ TE }$
Tatsächlicher Klassifikationsfehler (true classification error)	$F_{TE}(K) = \frac{ \{o \in TE \mid K(o) \neq C(o)\} }{ TE }$
Beobachteter Klassifikationsfehler (apparent classification error)	$F_{TR}(K) = \frac{ \{o \in TR \mid K(o) \neq C(o)\} }{ TR }$

4.1.4. Precision und Recall

- Betrachte eine bestimmte Klasse c in einem Klassifikationsproblem
- Dann gibt es vier Möglichkeiten

	Vorhersage = c	Vorhersage = nicht c
Echte Klasse = c	True positive	False negative
Echte Klasse = nicht c	False positive	True negative

Precision	$\frac{\text{true positive}}{(\text{true positive}) + \text{false positive}}$	„Wenn die Klasse vorhergesagt wird, wie sicher ist die Vorhersage“
Recall	$\frac{\text{true positive}}{(\text{true positive}) + (\text{false negative})}$	„Wie oft wird die Klasse c wieder gefunden“

4.2. Bayes- Klassifikatoren**4.2.1. Motivation:**

- Gegeben: Ein Objekt o , zwei Klassen positiv und negativ, drei unabhängige Hypothesen h_1, h_2, h_3
- Die A – posteriorie Wahrscheinlichkeiten der Hypothesen für gegebenes o :
 $P(h_1|o) = 0,4$; $P(h_2|o) = 0,3$; $P(h_3|o) = 0,3$
- Die A- posteriori- Wahrscheinlichkeiten der Klassen für gegebene Hypothese:
 $P(\text{negativ}|h_1) = 0$, $P(\text{positiv}|h_1) = 1$; $P(\text{negativ}|h_2) = 1$, $P(\text{positiv}|h_2) = 0$;
 $P(\text{negativ}|h_3) = 1$, $P(\text{positiv}|h_3) = 0$
→ o ist mit Wsk. 0,4 positiv, mit Wahrscheinlichkeit 0,6 negativ

4.2.2. Optimaler Bayes- Klassifikator

- Sei $H = \{h_1, \dots, h_l\}$ eine Menge unabhängiger Hypothesen und o ein klassifizierendes Objekt
- Der optimale Bayes- Klassifikator ordnet o der folgenden Klasse zu: $\operatorname{argmax}_{c_j \in C} \sum_{h_i \in H} P(c_j|h_i) * P(h_i|o)$

Vereinfachung	immer genau eine der Hypothesen h_i gültig
vereinfachte Entscheidungsregel	$\operatorname{argmax}_{c_j \in C} P(c_j o)$ Umformung mit Satz von Bayes $\operatorname{argmax}_{c_j \in C} P(c_j o) = \operatorname{argmax}_{c_j \in C} \frac{P(o c_j) * P(c_j)}{P(o)} = \max_{c_j \in C} P(o c_j) * P(c_j)$
Maximum- Likelihood- Klassifikator	$\operatorname{argmax}_{c_j \in C} P(o c_j) * P(c_j)$ Endgültige Entscheidungsregel des optimalen Bayes-Klassifikators

4.2.3. Naiver Bayes- Klassifikator

- Schätzung der $P(c_j)$ als beobachtete Häufigkeit der einzelnen Klassen; Schätzung der $P(o|c_j)$
- Annahmen des naiven Bayes- Klassifikators:
 - $o = (o_1, \dots, o_d)$ und die Attribute o_i sind für eine gegebene Klasse bedingt unabhängig
- Entscheidungsregel des naiven Bayes- Klassifikators: $\operatorname{argmax}_{c_j \in C} P(c_j) * \prod_{i=1}^d P(o_i|c_j)$

4.2.4. Laplace Korrektur

- Problem: Bei kleinen Datensätzen tauchen oft 0-Wahrscheinlichkeiten auf
- Lösung: Laplace Korrektur
 - Rechne nicht mit der Wahrscheinlichkeit 0, sondern ergänze beim „Aufzählen“ der Wahrscheinlichkeiten einen Fall für jeden Attributwert

4.2.5. Diskussion: Bayes- Klassifikator

- Optimaler Bayes Klassifikator: Theoretische Optimalitätseigenschaft (→ Standard zum Vergleich für andere Klassifikatoren)
 - o Erforderliche Wahrscheinlichkeiten sind oft unbekannt
- Naiver Bayes Klassifikator:

Vorteile	Nachteile
Leicht zu implementieren	Annahme: Unabhängigkeit der Attribute
Schnelle Klassifikation	
Ausreichende Klassifikationsgenauigkeit in vielen Anwendungen	
Inkrementalität: Neue Trainingsobjekte können leicht eingearbeitet werden	

4.2.6. Anwendung Textklassifikation:

- Vokabular $T = \{t_1, \dots, t_d\}$ von relevanten Themen
- Repräsentation eines Textdokuments $o = \{o_1, \dots, o_d\}$ o_i : u. a. Häufigkeit des Auftretens von t_i in o
- Methode: Auswahl der relevanten Terme, Berechnung der Termhäufigkeit, Klassifikation neuer Dokumente
- Vorverarbeitung der auftretenden Worte auf Grundformen (Stemming)
- Auswahl geeigneter Terme:
 - o Eliminiere Stopwörter (und, der, die, das, ...) und Wörter die sehr selten auftauchen
 - o Wähle Terme aus, die gut zwischen Klassen trennen können
- Lernen des Naive Bayes- Klassifikators (Laplace-Korrektur)
- Anwenden des Klassifikators auf neue Dokumente
- Bei der Email-Spamfilterung
 - o Fälschliches Aussortieren einer Email ist viel schlimmer als fälschliches nicht Aussortieren
→ Sortiere nur aus, wenn die Wsk. für Spam um Faktor Q (z.B. $Q=10$) höher ist als für Nicht-Spam
 - o Wenn Nutzer Nachrichten als Spam/ nicht-Spam markieren, können diese Dokumente zum inkrementellen Update der Wsk. mit Klassifikator genutzt werden (→ inkrementelles Lernen)

4.3. Nächste- Nachbarn- Klasifikation

- „Lazy Leaner“:
 - o Es wird kein explizites Modell gelernt → Klassifikation erfolgt direkt auf den Trainingsbeispielen
- Analogielernen:
 - o Finde die ähnlichsten Trainingsbeispiele („Entscheidungsmenge“), klassifiziere das neue Beispiel genauso wie die Mehrheit dieser Beispiele
- Distanzfunktion zum Bestimmen der Ähnlichkeit
- Parameter k
 - o „zu kleines“ k : verallgemeinert zu wenig: hohe Sensitivität gegenüber Ausreißern
 - o „zu großes“ k : verallgemeinert zu viel: viele Objekte aus anderen Clustern in der Entscheidungsmenge
 - o mittleres k : höchste Klassifikationsgüte, oft $1 < k < 10$
- Parameterwahl
 - o Probiere verschiedene Parameter: $k = 1, 2, 3, \dots$
 - o Für jedes k : Ermittle Klassifikationsfehler: entweder auf der Trainingsmenge oder durch (k -fache) Kreuzvalidierung → Verwende bestes k zur Klassifikation

4.3.1. Entscheidungsregel

- Standardregel: Wähle die Mehrheitsklasse der Entscheidungsmenge
- Gewichtete Entscheidungsregel: Gewichte die Trainingsbeispiele in der Entscheidungsmenge

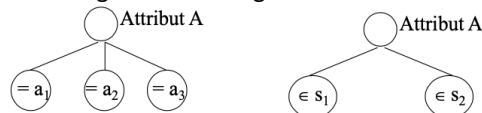

Vorteil	Nachteile
Erfordert als Eingabe nur Trainingsdaten	Ineffizient während der Klassifikation
Hohe Klassifikationsgenauigkeit in vielen Anwendungen	Liefert kein explizites Wissen über die Klassen
Inkrementalität: Modell muss nicht angepasst werden	
Varianten auch für die Vorhersage numerischer Werte einsetzbar	

4.4. Entscheidungsbaumlernen

- Eigenschaften:
 - o innerer Knoten repräsentiert ein Attribut
 - o Eine Kante repräsentiert einen Test auf dem Attribut des Vaterknotens
 - o Ein Blatt repräsentiert eine der Klassen
- Konstruktion: Top-Down anhand der Trainingsmenge
- Anwendung:
 - o Durchlauf des Entscheidungsbaums von der Wurzel zu einem der Blätter (Eindeutiger Pfad)
 - o Zuordnung des Objekts zur Klasse des erreichten Blatts

4.4.1. Konstruktion eines Entscheidungsbaumes

- Basis Algorithmus:
 - o Anfangs gehören alle Trainingsdaten zur Wurzel
 - o Auswahl des nächsten Attributes (**Splitstrategie**)
 - o Die Trainingsdatensätze werden unter Nutzung des Splitattributs partitioniert (d.h.: Jeder innere Knoten bezieht sich auf eine Teilmenge der Daten mit bestimmten Eigenschaften)
 - o Das Verfahren wird rekursiv für die Partitionen fortgesetzt
 - Lokal optimierender Algorithmus
 - Findet keinen global optimalen (= kleinsten) Baum
- Abbruchbedingungen:
 - o Keine weiteren Splitattribute
 - o Alle Trainingsdatensätze eines Blatts gehören zu selben Klasse
- Splitstrategien

Kategorische Attribute (Splitbedingung der Form Attribut = a; Attribut ∈ set)	<ul style="list-style-type: none"> - Eine mögliche Teilmenge 
Numerische Attribute (Splitbedingung der Form Attribut < a)	<ul style="list-style-type: none"> - Viele mögliche Splitpunkte 

Gegeben	<ul style="list-style-type: none"> - Eine Menge T von Trainingsobjekten mit disjunkter, vollständiger Partitionierung T_1, T_2, \dots, T_m - p_i die relative Häufigkeit der Klasse c_i in T
Gesucht	<ul style="list-style-type: none"> - Ein Maß der Unreinheit einer Menge S von Trainingsobjekten in Bezug auf die Klassenzugehörigkeit - Ein Split von T in T_1, T_2, \dots, T_m der dieses Maß der Unreinheit minimiert

4.4.2. Informationsgewinn

- **Entropie:** Minimale Anzahl von Bits zum Codieren der Nachricht, mit der man die Klasse eines zufälligen Trainingsobjekts mitteilen möchte
- $\text{entropie}(T) = -\sum_{i=1}^k p_i \log(p_i)$ (p_i ist die Wsk. für die Klasse i in T)
 $\text{entropie}(T) = 0$, falls $p_i = 1$ für ein i
 $\text{entropie}(T) = \log k$ (bei Gleichverteilung)
- Attribut A habe die Partitionierung T_1, T_2, \dots, T_m erzeugt
- Der Informationsgewinn des Attributs A in Bezug auf Menge T von Trainingsdaten ist definiert als:
 $\text{informationsGewinn}(T, A) = \text{entropie}(T) - \sum_{i=1}^m \frac{|T_i|}{|T|} * \text{entropie}(T_i)$

4.4.3. Gini- Index

- Gini- Index für eine Menge T von Trainingsobjekten: $\text{gini}(T) = 1 - \sum_{j=1}^k p_j^2$ ($\text{gini}(T) \in [0,1]$)
 - o Kleiner Gini- Index ↔ geringe Unreinheit
 - o Großer Gini- Index ↔ hohe Unreinheit
- Das Attribut A habe die Partitionierung T_1, T_2, \dots, T_m erzeugt
- Gini- Index des Attributs A in Bezug auf T ist definiert als $\text{gini}_A(T) = \sum_{i=1}^m \frac{|T_i|}{|T|} * \text{gini}(T_i)$
- Gewichtet größere Klassen stärker als Infogewinn und ist etwas effizienter

4.4.4. Ansätze zum Vermeiden von Overfitting- Überanpassung

- Entfernen von fehlerhaften, widersprüchlichen Trainingsdaten
- Wahl einer geeigneten Größe der Trainingsmenge (Nicht zu klein, nicht zu groß)
- Wahl geeigneter Größe des minimum support (Anzahl der Datensätze, die mindestens zu einem Blattknoten des Baum gehören müssen) → minimum support $\gg 1$
- Wahl geeigneter Größe der minimum confidence: (Anteil den die Mehrheitsklasse eines Blattknotens besitzen muss) → minimum confidence $\ll 100\%$
 - o Die Partitionen der Testdaten, die den Blättern zugeordnet sind, müssen dann nicht rein sein und können auch fehlerhafte Datensätze oder Rauschen enthalten
- Nachträgliches Pruning des Entscheidungsbaumes: Abschneiden überspezialisierter Äste
- **Fehlerreduktions- Pruning**
 - o Aufteilung der Trainingsdaten in kleinere Trainingsmenge und „Validierungsmenge“
 - o Konstruktion eines Entscheidungsbaums E für die Trainingsmenge
 - o Pruning von E mit Hilfe der Validierungsmenge V
 - Bestimme Teilbaum von E, dessen Abschneiden den Klassifikationsfehler auf V am stärksten reduziert → Entferne diesen Teilbaum → Fertig, falls kein solcher Teilbaum mehr existiert
 - o Nur anwendbar, wenn genügend viele klassifizierte Daten vorhanden
- **Kostenkomplexitäts- Pruning (KP)**
 - o Benötigt keine separate Validierungsmenge → auch anwendbar für kleine Datenmengen
 - o Pruning des Entscheidungsbaums mit Hilfe der Trainingsmenge
 - Klassifikationsfehler ungeeignet als Qualitätskriterium, da er auf der Trainingsmenge mit der Größe der Bäume monoton sinkt
 - o Neues Qualitätskriterium für Entscheidungsbäume:
 - Trade-off von Klassifikationsfehler und Baumgröße
 - Gewichtete Summe aus Klassifikationsfehler und Baumgröße
 - Intuition: Kleinere Entscheidungsbäume generalisieren typischerweise besser
- **Grundbegriffe (KP)**

$ E $	Größe des Entscheidungsbaums (Anzahl der Blätter)
$KK_T(E, \alpha)$	Kostenkomplexität von E in Bezug auf die Trainingsmenge T und den Komplexitätsparameter $\alpha \geq 0$: $KK_T(E, \alpha) = F_T(E) + \alpha * E $
kleinste minimierende Teilbaum $E(\alpha)$ Eigenschaften:	(1) $\forall B \leq E: KK_T(E(\alpha), \alpha) \leq KK_T(B, \alpha)$ (2) $\forall B \leq E, B \neq E(\alpha): (KK_T(E(\alpha), \alpha) = KK_T(B, \alpha)) \rightarrow E(\alpha) < B $ - $\alpha = 0$: E selbst besitzt die minimale Kostenkomplexität - $\alpha \rightarrow \infty$: Die Wurzel von E, minimiert die Kostenkomplexität - $0 < \alpha < \infty$: Ein echter Teilbaum (mehr als die Wurzel) hat minimale Kostenkomplexität
E_e	Teilbaum mit der Wurzel e
$\{e\}$	Baum, der nur aus dem Knoten e besteht
Weitere Eigenschaften	- Für kleine Werte von α gilt: $KK_T(E_e, \alpha) < KK_T(\{e\}, \alpha)$ - Für große Werte von α gilt: $KK_T(E_e, \alpha) > KK_T(\{e\}, \alpha)$ - Kritischer Wert von α für e <ul style="list-style-type: none"> ▪ $\alpha_{crit}: KK_T(E_e, \alpha_{crit}) = KK_T(\{e\}, \alpha_{crit})$ ▪ Für $\alpha \geq \alpha_{crit}$ lohnt sich das Prunen beim Knoten e
Schwächster Link	Knoten mit dem minimalen Wert für α_{crit}

- **Methode (KP)**
 - o Beginne mit dem vollständigen Baum E
 - o Entferne iterative immer schwächsten Link aus dem aktuellen Baum (ggf. mehrere im gleichzeitig)
 - o Folge ge- prune- ter Bäume $E(\alpha_1) > E(\alpha_2) > \dots > E(\alpha_m)$ mit $\alpha_1 < \alpha_2 < \dots < \alpha_m$
 - o Auswahl des besten $E(\alpha_i)$
 - Schätzung des Klassifikationsfehlers auf der Grundgesamtheit durch k- fache Überkreuz-Validierung auf der Trainingsmenge

- Interpretation als Regelmenge

- Jeder Entscheidungsbaum kann als eine Regelmenge interpretiert werden
 - Regelrumpf (linke Seite):
 - Konjunktion von Bedingungen über Attributen
 - Jede Bedingung ist ein Attribut- Wert- Paar
 - Regelkopf (rechte Seite): Ein Wert des Klassenattributes
- Für jeden Fall trifft genau eine Regel zu

4.5. Regellernverfahren

- Grundidee: Entscheidungsbäume können als Regelmenge interpretiert werden:
 - Regellernverfahren lernen die Sequenz von Regeln S („decision list“)
 - Angewandte Strategie beim Regellernen: „Sequential Covering“
 - Lernen eine Regel nach der anderen aus den Trainingsdaten
 - Suche die jeweils beste Regel für den verbleibenden Trainingsdatensatz
 - Eine Regel deckt ein Beispiel ab, wenn das Beispiel die Regelbedingung erfüllt
 - Alle Beispiele, die von der Regel abgedeckt werden, werden aus den Trainingsdaten entfernt, neue Regeln werden nur aus den verbleibenden Daten gelernt
 - Lernen weitere Regeln, bis alle Beispiele abgedeckt sind
 - Gehe die gelernte Sequenz S von Regeln der Reihe nach durch
 - Sobald eine Regelbedingung zutrifft, klassifiziere das Beispiel entsprechend der rechten Seite dieser Regel

4.5.1. Lernen einer einzelnen Regel

- Jede Konjunktion von Attribut- Wert- Paaren ist ein Kandidat
- Jeder Kandidat erhält eine Bewertung, wie gut er zur Klassifikation geeignet ist
- Greedy- Verfahren
 - Suche „beste“ Regel mit einer einzigen Bedingung
 - Erweitere diese Regel mit einer weiteren Bedingung
 - Wiederhole Erweiterung, bis keine Verbesserung mehr eintritt
 - Gib die Regel mit der besten Bewertung zurück
- Pruning bei Regellernverfahren
 - Problem: Overfitting
 - Divide- and- Conquer vs. Separate- and – Conquer
 - Bei Entscheidungsbäumen: Divide- and – Conquer
Pruning in einem Ast beeinflusst andere Äste nicht
 - Bei Regellernverfahren: Separate- and- Conquer
Pruning einer Regel beeinflusst alle weiteren Regeln
 - Idee: Vermeide Overfitting schon beim Erzeugen der einzelnen Regeln

4.5.2. IREP (Incremental Reduced Error Pruning) / Ripper Algorithmen

- Teile Trainingsdaten in 2 Teilmengen auf:
 - „Growing“- Menge ($\frac{2}{3}$ der Trainingsdaten)
 - „Pruning“- Menge ($\frac{1}{3}$ der Trainingsdaten)
- Jede Regel wird in 2 Phasen gelernt:

1. Phase	Growing: Füge solange Bedingungen hinzu, bis in dieser Menge keine negativen Beispiele mehr abgedeckt sind
2. Phase	Pruning: Entferne solange Bedingungen, bis in dieser Menge ein Gütemaß optimiert wird Beispiel Gütemaß: $x = \frac{\text{positive Beispiele}}{(\text{positive Beispiele}) + (\text{negative Beispiele})}$

- Stop- Kriterium: Gütemaß für neue Regeln ist unter einem bestimmten Wert

4.5.3. Classification by Association (CBA)

- Bei Sequential- Covering Regellernverfahren oder Entscheidungsbäumen: Entscheidungen hängen nur noch von wenigsten Instanzen ab → Overfitting
- Alternative zu Sequential Covering Verfahren: Nutze Assoziationsregeln für Klassifikation
- Normalerweise: Rechte Seite für Assoziationsregel variiert
- „Class Association Rules“ (CAR):
 - o Assoziationsregeln, bei denen das Klassenattribut auf der rechten Seite steht
- Regelpriorität
 - o Mehrere Regeln könne auf eine neue Instanz zutreffen
→ Sortiere Regeln geeignet (nach Vorrang >)
 - o Gegeben zwei Regeln R_1 und R_2 , dann hat R_1 Vorrang vor R_2 ($R_1 \succ R_2$), wenn
 - Die Konfidenz von R_1 größer ist als die Konfidenz von R_2
 - Die Konfidenz von R_1 gleich der Konfidenz von R_2 , aber der Support von R_1 größer ist
 - o Neue Trainingsbeispiele werden so klassifiziert, wie es die erste zutreffende Regel bestimmt
 - o Alternative zur Priorisierung: Abstimmung (Voting)

4.6. Support Vector Machines

4.6.1. Die optimale Hyperebene:

Lineare Trennbarkeit	Es existiert eine Hyperebene H, die die positiven und negativen Beispiele voneinander trennt
Optimale Hyperebene	H heißt optimale Hyperebene, wenn ihr Abstand d zum nächsten positiven und zum nächsten negativen Beispiel maximal ist

- Satz: Für eine linear trennbare Beispielmenge existiert eine eindeutig bestimmte optimale Hyperebene
- Ziel: Finde die optimale Hyperebene und nutze sie zur Klassifikation

Grundbegriffe:

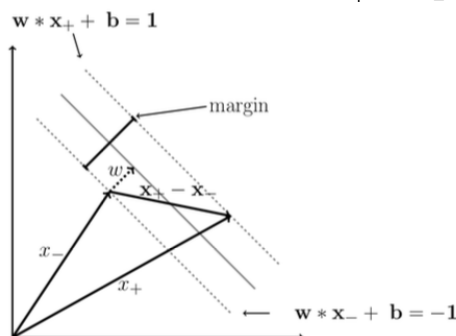
- o Gegeben: n Beispiele $(x_1, y_1), \dots, (x_n, y_n)$
 - x_i : Vektor der beschreibenden Attribute (nicht- Klassen- Attribute) im \mathbb{R}^n
 - y_i : Klasse des Beispiels: $y_i = \begin{cases} 1, & \text{falls Beispiele positiv} \\ -1, & \text{falls Beispiele negativ} \end{cases}$
- o Annahme: Beispiele sind linear trennbar
- o Jede Hyperebene wird beschrieben durch eine Funktion $f(x) = w * x + b = 0$
 - w : Gewichtsvektor; Normalenvektor (orthogonal auf Ebene) der Ebene
 - $w * x$: Skalarprodukt zwischen w und x
- o Ziel: Bestimme w und b so, dass gilt:
 1. $w * x + b \begin{cases} > 0, & \text{falls } x \text{ im positiven Halbraum} \\ = 0, & \text{falls } x \text{ auf } H \\ < 0, & \text{falls } x \text{ im negativen Halbraum} \end{cases}$
 2. $|w * x + b| = 1$ für alle Beispiele, die am nächsten zur Hyperebene liegen

Margins für linear trennbare Beispiele:

- o Definiere Hyperebenen H_1 und H_2 als parallele „Ränder“ des jeweils positiven/ negativen Bereichs:
 - $H_1: w * x_i + b \geq 1$ bei $y = +1$
 - $H_2: w * x_i + b \leq -1$ bei $y = -1$

$$y_i(w * x_i + b) - 1 \geq 0$$

- o Betrachte die beiden Vektoren x_+ und x_- :



- laut Definition:

$$w * x_+ = 1 - b \quad w * x_- = -1 - b$$

- Abstand zw. x_+ und H_2 :
 $margin = \frac{w}{\|w\|} * (x_+ - x_-)$

- somit ist der Margin:

$$margin = \frac{2}{\|w\|}$$

- **Maximaler Margin:**
 - H_1 und H_2 liegen parallel, haben also denselben Normalenvektor w
 - Per Konstruktion liegt kein Beispiel zwischen H_1 und H_2
 - Gesucht ist die Hyperebene H mit dem maximalen Margin
 - Um $\frac{2}{||w||}$ zu maximieren, muss man $||w||$ minimieren, unter den Nebenbedingungen $y_i(w * x_i + b) - 1 \geq 0$ für alle Beispiele $i = 1, \dots, n$
- **SVM Klassifikator**
 - Primales Optimierungsproblem:
Minimiere: $\frac{1}{2} ||w||^2$
Unter den Nebenbedingungen $y_i(w * x_i + b) - 1 \geq 0$
 - Konvexes, quadratisches Optimierungsproblem \rightarrow in $O(n^3)$ lösbar
 - Durch Minimierung von $\frac{1}{2} ||w||^2$ findet man eine eindeutige Hyperebene, die die beiden Halbräume am weitesten trennt
 - Aus der Nebenbedingung folgt außerdem: $y_i > 0 \leftrightarrow w * x_i + b > 0$
 - Daraus folgt kann man folgenden Klassifikation: $f(x) = w * x + b$
 - Die Klasse ist eindeutig durch $\text{sgn}(f(x))$ bestimmt

4.6.2. Berechnung der optimalen Hyperebene

- Momentan: Optimierungsproblem mit Nebenbedingungen (primales Problem)= Schwierig zu lösen
- **Lagrange- Funktion:** $L(x, \dots, z, \alpha) = f(x, \dots, z) - \sum_{i=1}^n \alpha_i g_i(x, \dots, z), \alpha_i \geq 0$
 - Problem: $\min_{x, \dots, z} f(x, \dots, z)$ wird zu $\min_{x, \dots, z} \max_{\alpha} L(x, \dots, z, \alpha)$
$$\min_{w, b} \max_{\alpha} L(w, b, \alpha) = \min_{w, b} \max_{\alpha} \frac{||w||^2}{2} - \sum_{i=1}^n \alpha_i (y_i(w * x_i + b) - 1), \alpha_i \geq 0$$
- **Karush- Kuhn- Tucker (KKT) Bedingungen:**
 - Für das Optimierungsproblem gelten die KKT Bedingungen gdw. x, \dots, z, α die Lösung des Problems sind. Für jeden Parameter x, \dots, z der Lagrange- Funktion gilt dann:
 - Stationarität: $\frac{\partial L(x, \dots, z, \alpha)}{\partial x} = 0, \dots, \frac{\partial L(x, \dots, z, \alpha)}{\partial z} = 0$
 - Primal feasibility: $g_i(x, \dots, z) \geq 0$
 - Dual feasibility: $\alpha_i \geq 0$
 - Complementary slackness: $\sum_{i=1}^n \alpha_i g_i(x, \dots, z, \alpha) = 0$
 - Hier:
 - Stationarität: $\frac{\partial L(w, \alpha)}{\partial w} = 0, \dots, \frac{\partial L(w, \alpha)}{\partial b} = 0$
 - Primal feasibility: $y_i(w * x_i + b) - 1 \geq 0$
 - Dual feasibility: $\alpha_i \geq 0$
 - Complementary slackness: $\alpha_i (y_i(w * x_i + b) - 1) = 0$

- SVM Optimierungsproblem – Duale Form

- Maximiere: $L(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{j=1}^n y_i y_j \alpha_i \alpha_j x_i * x_j$
unter den Nebenbedingungen $\forall i: 0 \leq \alpha_i$ und $\sum \alpha_i y_i = 0$
- Für jedes Trainingsbeispiel gibt es ein α_i in der Lösung
- Aus der Complementary Slackness (KKT Bedingungen) folgt:
 - $\alpha_i = 0$ heißt, dass das Beispiel x_i im passenden Halbraum liegt
 - $\alpha_i > 0$ heißt, dass x_i ein Stützvektor (Support Vector) ist
- Und mit $w = \sum \alpha_i y_i x_i$ folgt: beste Normalenvektor w ist eine Linearkombination von Stützvektoren
- Klassifikationsfunktion: $f(x) = \text{sgn}(\sum \alpha_i y_i x_i * x + b)$

4.6.3. Optimierungsalgorithmus: Sequential Minimal Optimization (SMO)

- Intuition: quadratisches Optimierungsproblem in zwei Variablen ist analytisch lösbar
- Optimierungsfunktion für zwei α s (α_a und α_b)

$$\max_{\alpha_a, \alpha_b} \alpha_a + \alpha_b - \frac{1}{2} (k_{a,a} \alpha_a^2 + k_{b,b} \alpha_b^2 + \alpha_a y_a \sum_{j \neq a} \alpha_j y_j k_{a,j} + \alpha_b y_b \sum_{j \neq b} \alpha_j y_j k_{b,j})$$
mit $k_{i,j} = x_i * x_j$ und $\alpha_a y_a + \alpha_b y_b = - \sum_{j \neq a, b} \alpha_j y_j$

- Ablauf:
 1. Initialize α s with 0
 2. Finde ein α_a , dass die KKT Bedingungen verletzt
 3. Wähle ein zweites α_b , mit einer Heuristik
 4. Optimize für das Paar, alle anderen α sind fix
 5. Wiederhole Schritte 2. Bis 4. Bis zur Konvergenz
- Andere Optimierungsalgorithmen versuchen auf folgende Arten das Problem effizient zu lösen:
 - Auswahl des „working sets“
 - Trennung der Parameter in:
 - Fixe Parameter, die nicht geupdated werden (Menge N)
 - Parameter, die geupdated werden (Menge B)
 - Führe das Optimierungsproblem auf Menge B aus
 - Shrinking des Optimierungsproblems
 - Für viele Tasks ist die Anzahl der Support Vektoren viel kleiner als die Anzahl der Trainingsbeispiele
 - Wären die SVs apriori bekannt, wäre es hinreichend die SVM auf diesen zu trainieren
 - Während dem Training wird sehr schnell klar, dass bestimmte Trainingsbeispiele niemals SVs werden → Streiche diese aus dem Optimierungsproblem (= verkleinere das Optimierungsproblem)

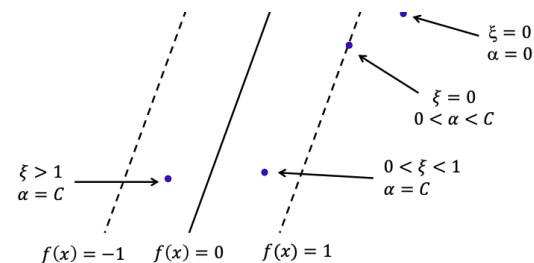
4.6.4. Zusammenfassung SVM bei linear trennbaren Daten

- Maximierung des Margins einer Hyperebene ergibt eine eindeutige Festlegung der optimalen trennenden Hyperebene
- Maximierung des Margins \leftrightarrow Minimierung der Länge des Normalenvektors w unter Nebenbedingungen
- Berechnung des Normalenvektors als Optimierungsproblem mit Hilfe z.B. des Sequential Minimal Optimization Algorithmus
- Das Lernergebnis ist eine Linearkombination von Stützvektoren der Hyperebene
- Der Klassifikationsprozess beschränkt sich also nur noch auf das Ausrechnen einiger Skalarprodukte (linearer Aufwand bzgl. der Anzahl der Stützvektoren)

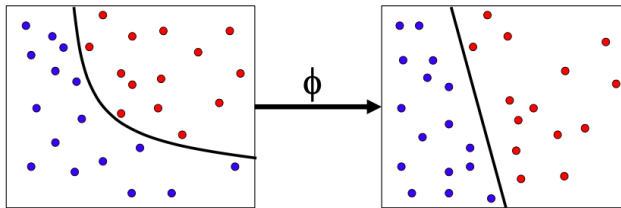
4.6.5. Nicht linear trennbare Daten

- Möglicher Ansatz: Entferne eine minimale Menge von Datenpunkten, so dass die Daten linear trennbar werden (minimale Fehlklassifikation) → Problem: Komplexität des Algorithmus wird exponentiell
- Bessere Lösungen: „weich“ trennende Hyperebene, Kernelmethoden
- Weich trennende Hyperebene:

- Praktikable Lösung, wenn Daten „fast“ linear trennbar sind
- Primales Optimierungsproblem:
 Minimiere: $\frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i$
 Unter den Nebenbedingungen $y_i(w * x_i + b) - 1 + \xi_i \geq 0$ und $\xi_i \geq 0$
- Parameter $C \in \mathbb{R}, C > 0$
 - Je größer der Parameter C , desto stärker werden Fehler gewichtet
 - Wahl oft durch eine (exponentielle) Rastersuche und Überkreuzvalidierung:
 Teste $C = 2^{-5}, C = 2^{-4}, \dots, C = 2^1 \dots C = 2^{13}$
- Schlupfvariablen („Slack variables“) ξ_i :
 - Lassen Fehler beim Trennen zu
 - Verringern möglicherweise die Anzahl der Stützvektoren
 - Berechnung:
 - $f(x_i) = w * x_i + b \geq 1 - \xi_i$ für $y_i = 1$ und
 - $f(x_i) = w * x_i + b \leq -1 + \xi_i$ für $y_i = -1$
- Duales Optimierungsproblem: $\sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n y_i y_j \alpha_i \alpha_j x_i * x_j$
 s. t. $\forall i: 0 \leq \alpha_i \leq C$ und $\sum \alpha_i y_i = 0$



- Kernel Methoden:



- Idee: Erreiche Lineare Trennbarkeit durch Transformation ϕ der Daten in einen anderen Raum
 1. Nicht- lineare Einbettung in einen (höher- dimensional) Raum
 2. Dort Suchen der optimal trennenden Hyperebene
 3. Rücktransformation

- Kernel – Funktionen

- Die Anzahl der Dimensionen im Feature Space kann sehr groß werden
- Berechnungen werden sehr aufwendig (oft: zu aufwendig)
- Lösung: Kernel- Funktion
 - Berechne die Datenpunkte im hochdimensionalen Raum nicht explizit
 - Formulierung des Dualen Problems benötigt nur das Skalarprodukt für zwei Punkte
→ Die Datenpunkte selbst werden gar nicht benötigt
 - Verwende Kernelfunktionen $K: K(x, y) = \phi(x) * \phi(y)$
Eine Kernelfunktion entspricht einem Skalarprodukt zweier Punkte in einem höherdimensionalen Raum
- Typische Kernelfunktionen:

Linearer Kernel	$K(x_i, x_j) = x_i * x_j$
Polynomial- Kernel	$K(x_i, x_j) = (x_i * x_j + 1)^h$ für $h \in \mathbb{N}$
RBF- Kernel	$K(x_i, x_j) = e^{-\frac{\ x_i - x_j\ ^2}{\sigma^2}}$ (oft auch: $e^{-\gamma \ x_i - x_j\ ^2}$, mit $\gamma = \frac{1}{\sigma^2}$)
Sigmoidkernel	$K(x_i, x_j) = \tanh(\kappa x_i * x_j - \delta)$

- Optimierungs Problem mit Soft Margin und Kernen

Primal optimization problem	Dual optimization problem
Minimize: $\frac{1}{2} \ w\ ^2 + C \sum_{i=1}^n \xi_i$ Subject to $y_i(w * \phi(x_i) + b) - 1 + \xi_i \geq 0$ and $\xi_i \geq 0$	Minimize $\sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n y_i y_j \alpha_i \alpha_j K(x_i, x_j)$ Subject to $0 \leq \alpha_i \leq C$ and $\sum \alpha_i y_i = 0$

- Klassifikationsfunktion $f(x) = \text{sgn}(\sum \alpha_i y_i K(x_i, x) + b)$

Vorteile SVM	Nachteile SVM
Eines der momentan besten/ beliebtesten Klassifikationsverfahren	Funktioniert nur bei numerischen Featurewerten (->Spezielle Erweiterungen)
Bei gelerntem Modell schnelle Klassifikation	Funktioniert nur für zwei Klassen Probleme (->Spezielle Erweiterungen)
	Blackbox: Trennende Hyperebene ist nur schwer für Menschen zu verstehen

4.7. Kombination von Klassifikationsverfahren

- Grundidee: Klassifikationen kombinieren, um die Klassifikationsgüte zu verbessern
- Zwei unterschiedliche Ansätze: Bagging, Boosting

4.7.1. Bagging- Bootstrap Aggregating

- Meta- Verfahren: Schleife um einen beliebigen anderen Klassifikator
- Bootstrapping- Schritt
 - Gegeben: Trainingsdatensatz mit n Beispielen
 - Ziehe aus dem Trainingssatz m Samples (zufällig, mit Zurücklegen) der Größe n
 - Für großen n enthält jedes Sample dann $\approx 37\%$ Duplikate

- Training: Lerne ein eigenes Klassifikationsmodell für jedes Sample (mit dem selben Klassifikator)
- Testen
 - o Für jedes neu zu klassifizierende Beispiel befrage jedes Klassifikationsmodell
 - o Stimme ab („Voting“), Mehrheitsentscheidung bestimmt die Vorhersage

Vorteile	Nachteile
Bagging führt zu einer verbesserten Klassifikationsgüte bei instabilen Klassifikationsmethoden (falls das Modell stark abhängig von der Stichprobe)	Leichte Einbußen bei der Klassifikationsgüte bei stabilen Modellen
Vermeidet Overfitting	Modelle sind nicht mehr leicht interpretierbar
	Vergrößerte Laufzeit (um linearen Faktor m)

- **Random Forest**
 - o Führe Bagging mit Entscheidungsbäumen durch
 - o Problem: Die Entscheidungsbäume sind alle ähnlich zueinander
 - o Idee: Erzwingen Unterschiede
 - o Bei jedem Split suche das Splitattribut nicht unter allen Attributen, sondern unter einer zufälligen Teilmenge der Attribute
 - Kleine Teilmenge: Schwächere Korrelation zwischen den Bäumen, schlechtere Vorhersagegenauigkeit
 - Große Teilmenge: Stärkere Korrelation zwischen den Bäumen, bessere Vorhersagegenauigkeit
 - Typisch: Bei k Attributen im Datensatz, verwende für jeden Split eine Teilmenge von \sqrt{k} Attributen
 - o Optionale Gewichtung:
 - Messe für jedes Klassifikationsmodell die Vorhersagegenauigkeit bei den nicht verwendeten Beispielen
 - Gewichte Abstimmung nach Genauigkeit: Bessere Modelle erhalten ein größeres Gewicht

4.7.2. Boosting

- Schwacher Lerner: Die Vorhersage eines Klassifikators ist nur schwach mit dem tatsächlichen Wert korreliert

Training	<ul style="list-style-type: none"> - Erzeuge eine Sequenz von Klassifikatoren mit dem gleichen Algorithmus - Jeder Klassifikator hängt vom vorherigen ab und konzentriert sich auf dessen Fehler - Beispiele, die im vorherigen Klassifikator falsch vorhergesagt wurden, werden vom nächsten Klassifikator höher gewichtet
Test:	<ul style="list-style-type: none"> - Für jeden Klassifikator bestimme Klassifikationsgüte - Endgültige Entscheidung: Abstimmung aller Klassifikatoren der Sequenz gewichtet nach Klassifikationsgüte

4.7.3. AdaBoost

- Alle Beispiele x_i erhalten initiales Gewicht von $w_1(i) = 1/n$
- T Klassifikatoren werden in T Iterationen gelernt
- In jeder Iteration t (1, 2, ..., T):
 - o Sampling eines Trainingssets: Beispiele werden gezogen mit Wsk. ihres Gewichtes, mit Zurücklegen
 - o Lerne Klassifikator auf dem Trainingsset (Model M_t)
 - o Fehler des Klassifikationsmodells: $\text{err}(M_t) = \sum_i w_t(i) * \text{err}(x_i)$
 - $\text{err}(x_i)$: Klassifikationsfehler für das Beispiel x_i
 - o Falls x_i richtig klassifiziert wurde, update Gewicht $w_{t+1}(i) = w_t(i)$
 - o Normalisiere Gewichte, so dass die Summe aller Gewichte 1 ergibt
(→ je höher ein Gewicht ist, desto schwieriger ist das Beispiel zu klassifizieren)
- Klassifikation neuer Beispiele
 - o Berechne Vorhersage aller Klassifikatoren
 - o Gewichtete Abstimmung aller Klassifikatoren
 - o Gewicht für jeden Klassifikator: $\log \frac{1-\text{err}(M_t)}{\text{err}(M_t)}$

4.7.4. Vergleich Bagging und Boosting

- Beide können die Klassifikationsgüte stark verbessern im Vergleich zu einem einzelnen Modell
- Boosting verbessert tendenziell stärker, konzentriert sich stark auf „schwierige“ Beispiele
→ Bagging ist weniger anfällig für Overfitting

5. Regressionsanalyse**5.1. Lineare Regression****5.1.1. Einführung**

- Problembeschreibung und Notation
 - o Gegeben: Eine Menge von Messungen x und eine abhängige Zielgröße y
 - o Ziel: Sage y mit Hilfe von x vorher
 - o Überwachtes Lernen (Trainingsmenge)
- Notation:

m	Anzahl von Trainingsbeispielen
x	„Eingabe“ Variable/ Merkmale
y	„Ausgabe“ Variable/ „Ziel“- Merkmale
(x,y)	Trainingsbeispiel
$(x^{(i)}, y^{(i)})$	i -tes Trainingsbeispiel
h	Bei linearer Regression mit einer Variablen (x): $h_{\theta}(x) = \theta_0 + \theta_1 x$

- **Kostenfunktion J**
 - o Idee: Wähle θ_0 und θ_1 so, dass $h_{\theta}(x)$ nahe bei y für alle Trainingsbeispiele (x,y) liegt
 - o Bestimmung des Fehlers durch eine Kostenfunktion J , die abhängig von den gewählten Parametern θ_0 und θ_1 ist: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m h_{\theta}(x^{(i)}) - y^{(i)}^2$
squared error function: $h_{\theta}(x^{(i)}) = \theta_0 + \theta_1 x^{(i)}$
 - o Lerne θ_0, θ_1 mit Hilfe der Kostenfunktion, indem der Lernalgorithmus $J(\theta_0, \theta_1)$ minimiert
→ minimiere $J(\theta_0, \theta_1)$

5.1.2. Gradient Descent

- Grundidee:
 - o Gegeben eine Funktion $J(\theta_0, \theta_1)$
 - o Ziel: minimiere $J(\theta_0, \theta_1)$
- Vorgehen
 - o Starte mit einer Belegung für θ_0 und θ_1
 - o Ändere θ_0 und θ_1 sodass $J(\theta_0, \theta_1)$ verringert wird, bis $J(\theta_0, \theta_1)$ nicht mehr minimiert wird

Gradient Descent Algorithmus

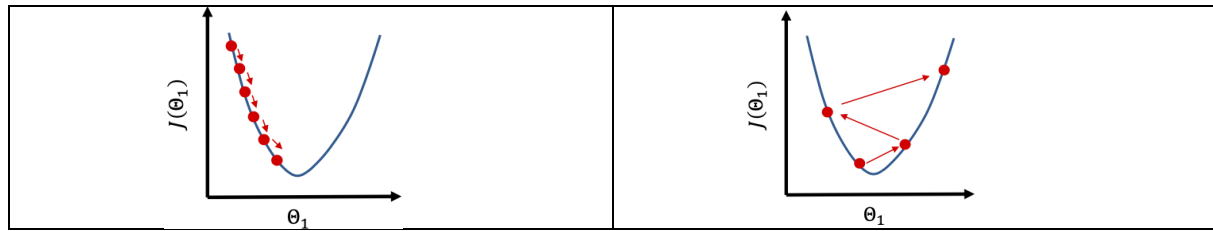
- o $\theta_1 = \theta_1 - \Delta\theta$ mit $\Delta\theta = \frac{\partial}{\partial \theta_1} J(\theta_1)$

repeat until convergence {	
$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$	für $j=0$, und $j=1$
}	α : Lernrate

- o θ_0 und θ_1 gleichzeitig updaten
 - $\text{temp}_0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$ $\theta_0 = \text{temp}_0$
 - $\text{temp}_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$ $\theta_1 = \text{temp}_1$
- Lernrate:
 - o $\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

Wenn α zu klein, kann Gradient Descent langsam werden

Wenn α zu groß, kann Gradient Descent über das Minimum springen.
Ggf. auch nicht konvergieren, ggf. auch divergieren



- Lokales vs. Globales Optimum
 - o Gradient Descent terminiert bei lokalem Minimum auch bei fixer Lernrate, da Steigung immer kleiner wird, je näher man an das lokale Minimum kommt

5.1.3. Gradient Descent und Lineare Regression

- Gradient Descent für Lineare Regression

```
repeat until convergence {
     $\theta_0 = \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m h_{\theta}(x^{(i)}) - y^{(i)}$ 
     $\theta_1 = \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m h_{\theta}(x^{(i)}) - y^{(i)}$ 
}
```

- „Batch“ Gradient Descent : Bei jedem Updateschritt werden alle Trainingsbeispiele verwendet
- Alternativ: „Stochastic“ Gradient Descent
 - o Für jedes Trainingsbeispiel update Parameter; verwende $(h_{\theta}(x^{(i)}) - y^{(i)})x^{(i)}$ statt Durchschnitt

5.1.4. Gradient Descent und Lineare Regression

- Notation
 - o n: Anzahl Merkmale
 - o $x^{(i)}$: Werte i-tes Trainingsbeispiel (jetzt Vektor)
 - o $x_j^{(i)}$: Wert Feature j des i-ten Trainingsbeispiels
- Hypothese: mehrere Merkmale: $h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$
Zur Vereinfachung der Notation, definiere $x_0 = 1$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \dots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

5.1.5. Andere Optimierungsverfahren: Normal Equation

- Wenn $\theta \in \mathbb{R}$ $J(\theta) = a\theta^2 + b\theta + c$
 - o Ableitung bilden und Minimum bestimmen $\frac{\partial}{\partial \theta} J(\theta) = \dots = 0$
 - o Löse nach θ
- Wenn $\theta \in \mathbb{R}^{n+1}$
 - o $\frac{\partial}{\partial \theta_j} J(\theta) = \dots = 0$ für jedes j
 - o Löse nach $\theta_0, \theta_1, \dots, \theta_n$
- Definiere Matrix X, Spalten sind die Merkmalsvektoren $X = (x_0 \ x_1 \ \dots \ x_n)$
- Kostenfunktion umschreiben (Summe als Matrixmultiplikation)
 $J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \rightarrow J(\theta) = \frac{1}{2m} (X\theta - y)^T (X\theta - y)$
- $\frac{1}{2m}$ später nicht wichtig, da später mit 0 verglichen wird
 $J(\theta) = ((X\theta)^T - y^T)(X\theta - y) = (X\theta)^T X\theta - (X\theta)^T y - y^T X\theta + y^T y$ $X\theta, y$ Vektor
 $J(\theta) = \theta^T X^T X\theta - 2(X\theta)^T y + y^T y$
 $\theta = (X^T X)^{-1} X^T y$

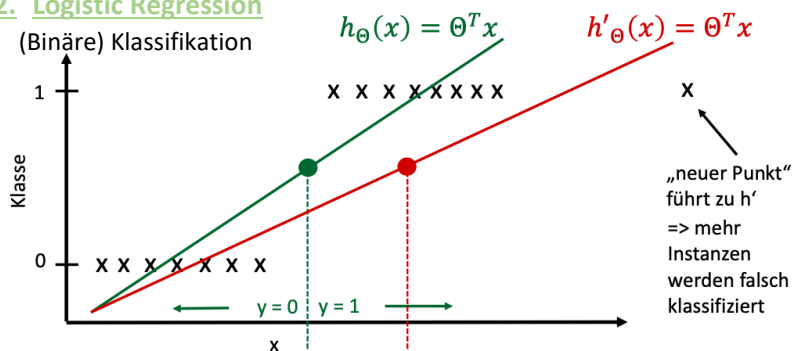
Gradient Descent	Normal Equation
α muss gewählt werden	α muss nicht gewählt werden
Iterationszahl groß	Keine Iteration
Funktioniert auch bei großem n	Muss $(X^T X)^{-1}$ berechnen ($n \times n$ Matrix; $O(n^3)$)
	Bei großem n langsam
	Funktioniert nicht bei Logistic Regression

- Andere Algorithmen zur Optimierung
 - o Conjugate Gradient
 - o Broyden- Fletcher- Goldfarb- Shanno (BFGS) Algorithmus
 - o Limited- memory BFGS (L- BFGS)

Vorteile	Nachteile
Kein α zu wählen	Komplexer
Meist schneller als Gradient Descent	

5.2. Logistic Regression

- (Binäre) Klassifikation



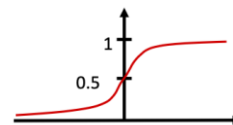
- o Schwellenwert- Klassifikator $h_\theta(x)$ bei 0,5
 - Wenn $h_\theta(x) \geq 0,5$, klassifiziere y als positiv
 - Wenn $h_\theta(x) < 0,5$, klassifiziere y als negativ

- **Logistic Regression**

- o Eigenschaft: $0 \leq h_\theta(x) \leq 1$
- o Logistic Regression: Klassifikationsalgorithmus

$g(z)$
logistic function
Sigmoid function

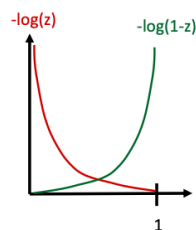
$$\left. \begin{aligned} h_\theta(x) &= g(\theta^T x) \\ g(z) &= \frac{1}{1+e^{-z}} \end{aligned} \right\} h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$



$$\begin{aligned} h_\theta(x) \geq 0.5 &\Leftrightarrow \theta^T x \geq 0 \\ h_\theta(x) < 0.5 &\Leftrightarrow \theta^T x < 0 \end{aligned}$$

5.2.1. Kostenfunktion:

- Linear Regression: $J(\theta) = \frac{1}{m} \sum_{i=1}^m \frac{1}{2} (h_\theta(x^{(i)}) - y^{(i)})^2$
 $J(\theta) = \frac{1}{m} \sum_{i=1}^m \text{cost}(h_\theta(x^{(i)}), y^{(i)})$
 $\text{cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)), & y = 1 \\ -\log(1 - h_\theta(x)), & y = 0 \end{cases}$



Falls $h_\theta(x) = 1 \Rightarrow \text{cost} = 0$
 Falls $h_\theta(x) = 0 \Rightarrow \text{cost} \rightarrow \infty$
 Falls $h_\theta(x) = 1 \Rightarrow \text{cost} \rightarrow \infty$
 Falls $h_\theta(x) = 0 \Rightarrow \text{cost} = 0$

- Umschreiben

- o Entfernen der Fallunterscheidung aus obiger Formel:
 $\text{cost}(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x))$
 - $y = 0$: $-0 \log(h_\theta(x)) - (1 - 0) \log(1 - h_\theta(x))$
 - $y = 1$: $-1 \log(h_\theta(x)) - (1 - 1) \log(1 - h_\theta(x))$

- Gradient Descent Logistic Regression

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right]$$

- o Allgemein:

repeat until convergence {
 $\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$
 }

- Logistic Regression:

repeat until convergence {
 $\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$
 }

5.3. Weitere Algorithmen

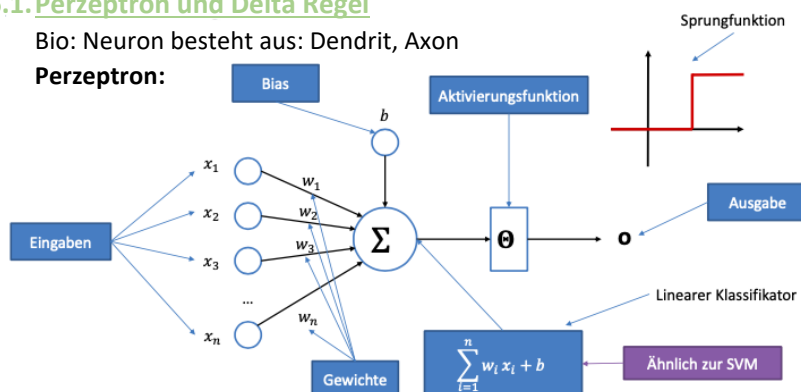
- Einführung:
 - o Regression = Klassifikation mit $|C| \rightarrow \infty$
 → Adaption von Klassifikationsalgorithmen für Regression

K-nearest neighbor	<ul style="list-style-type: none"> - Distanzfunktion für numerische Attribute - (Gewichteter) Mittelwert ersetzt Mehrheitsentscheid
Decision Trees	<ul style="list-style-type: none"> - Qualitätsmaß des Splits anpassen, z.B. <ul style="list-style-type: none"> ▪ Mittlerer quadratischer Fehler ▪ Mittlerer absoluter Fehler - Ausgabe: Mittelwert oder Median des Blattes
Support Vector Machines	<ul style="list-style-type: none"> - Basis: soft-margin SVM Klassifikation - Verwende Verlustfunktion für Regression Meistens: ϵ-insensitive Verlustfunktion - Primales Optimierungsproblem $\text{Minimize: } \frac{1}{2} \ w\ ^2 + C \sum_{i=1}^n \xi_i + \xi_i^*$ $\text{Subject to } y_i - w * x_i - b + \xi_i \leq \epsilon + \xi_i$ $w * x_i + b - y_i \leq \epsilon + \xi_i^* \text{ and } \xi_i, \xi_i^* \geq 0$ - Duales Optimierungsproblem $\text{Maximize: } \sum_{i=1}^n (\alpha_i^* - \alpha_i) y_i - \sum_{i=1}^n (\alpha_i^* - \alpha_i) \epsilon -$ $\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n (\alpha_i^* - \alpha_i) (\alpha_j^* - \alpha_j) x_i * x_j$ $\text{Subject to: } 0 \leq \alpha_i^* \leq C \text{ and } \sum (\alpha_i - \alpha_i^*) = 0$ - Regressionsfunktion: $f(x) = \sum_{i=1}^n (\alpha_i^* - \alpha_i) x_i * x + b$

6. Neuronale Netzwerke

6.1. Perzeptron und Delta Regel

- Bio: Neuron besteht aus: Dendrit, Axon
- **Perzeptron:**



- Bias „Trick“
 - o Zur Vereinfachung: Füge $x_0 = -1$ für den Bias mit dem Gewicht $w_0 = b$ ein
 → $\sum_{i=0}^n w_i x_i = w^T x$ (jetzt beginnt die Summe bei 0 statt 1)

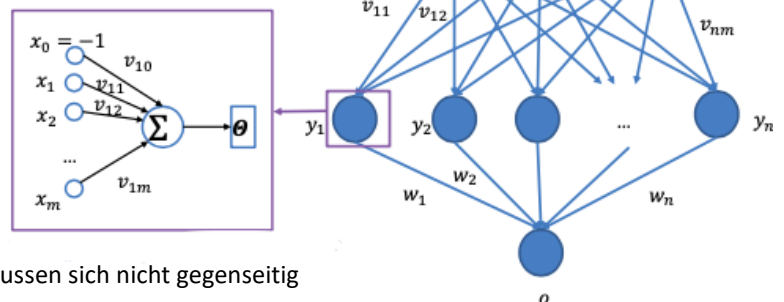
6.1.1. Delta Regel

- Datensatz: $(x; y)$
- Trainingsbeispiele ins Netz eingeben (überwachtes Lernen, da y bekannt)
- Vergleich Soll- mit Ist- Ausgabe → Bei Diskrepanz werden Gewichte angepasst
- Fehlerfunktion/ Kostenfunktion: $L(W) = \frac{1}{2|D|} \sum_{d \in D} (t_d - o_d)^2$
 - Minimiere die Kostenfunktion L -Gradient Descent
 - Für einen einzelnen Datensatz: $-\theta'(w^T x_d)(t_d - o_d)x_d$
- Vereinfachte Variante:
 - Stochastic Gradient Descent mit folgender Update Regel für die Gewichte:

$$w = w + \eta * x * (t - o)$$
 - t : Target (Soll-Ausgabe) o : Output (Ist Ausgabe)
 - + Gradient hatte negatives Vorzeichen
 - η Einheitliche Lernrate, statt $\alpha \theta'(w^T x)$
 - Setze Lernrate $\eta = 1$:
 1. Start: Initialisiere Gewichte zufällig
 2. Wähle zufällig einen Punkt (x, t) aus der Trainingsmenge:
 - a) Berechne $o = \theta(w^T, x)$
 - b) Falls $o > 0$ und $t = 0$:
 - Setze $w = w - x$
 - Falls $o \leq 0$ und $t = 1$:
 - Setze $w = w + x$
 3. Wiederhole 2 bis alle Punkte betrachtet wurden
 4. Falls sich die Gewichte geändert haben, gehe zu 2 (Starte eine neue Epoche)
- Konvergenz und Korrektheit:
 - Mit der Delta Regel kann ein Perzeptron in endlich vielen Schritten eine Klasseneinteilung gelernt werden, wenn es diese Einteilung lernen kann
 - Problem: Falls das Perzeptron nicht lernt, kann nicht unterschieden werden zwischen:
 - Es wurden noch nicht genügend Schritte beim Lernen vollzogen
 - Problem ist nicht mit einem Perzeptron lösbar
 - Hinweis: Es gibt keine obere Schranke für die Lerndauer
- XOR- Problem:
 - UND-Problem: linear separierbar
 - XOR- Problem: nicht linear separierbar: Einführung einer weiteren Hyperebene, so dass auch hier die Klassen getrennt werden können (Realisierung: weitere Zwischenschicht)

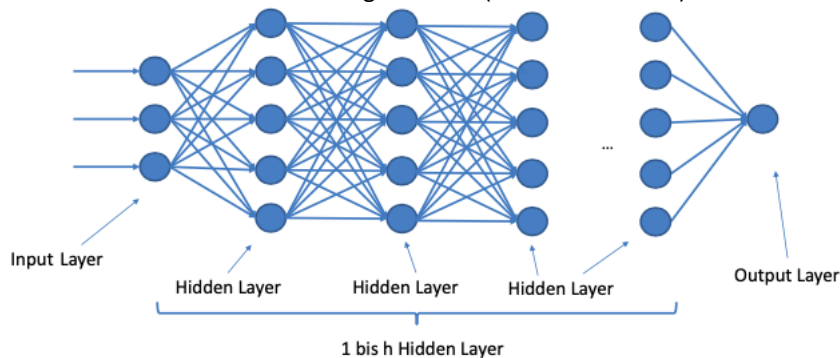
6.2. Multi-Layer Perzeptron und Backpropagation**2- Layer Perzeptron:**

- Input Vektor x
- Gewichtsmatrix V
- Aktivierungsvektor y
- Gewichtsmatrix W
- Output o
- $y = \theta(Vx)$
- $o = \theta(Wy)$
- Die Neuronen der zweiten Schicht beeinflussen sich nicht gegenseitig
- Können getrennt betrachtet werden
- Bei mehrschichtigen Netzen geht die Unabhängigkeit verloren

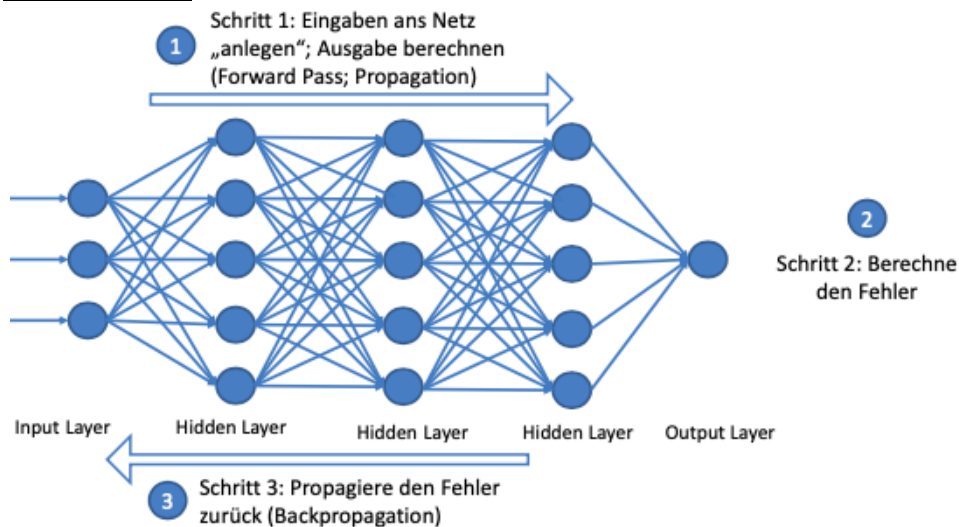


- Multi-Layer Perzeptron

- Jedes Neuron hat eine Aktivierungsfunktion (meist nicht linear)



6.2.1. Backpropagation



- Fehlerfunktion

- Fehler/ Kostenfunktion von der Delta Regel
- Wie schauen uns nun die Gradienten der einzelnen Schichten eines 2 Layer Perzeptrons an (immer nur bezogen auf eine Eingabe $d \in D$)

- Verallgemeinerung:

- Schrittweise Berechnung der Gradienten auch für mehr als zwei Schichten möglich
- Fehler wird dann schrittweise nach oben propagiert (Back- Propagation)

- Anwendung:

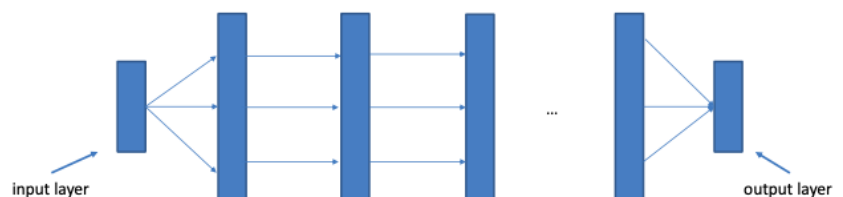
- Je öfter wie die Backpropagation durchführen, umso kleiner wird der Gradient (Probleme bei tieferen Schichten)
- Backpropagation Algorithmus lässt Abbruchbedingung offen
- Schlecht: Iteriere den Algo so lange bis der Fehler im Trainingsdatensatz unter einem vorgegebenen Schwellwert liegt -> führt zu Overfitting
→ Verwende separate Menge von Beispielen zum Validieren; iteriere solange, bis der Fehler dort minimal ist

6.3. Netzwerktypen

6.3.1. Basics

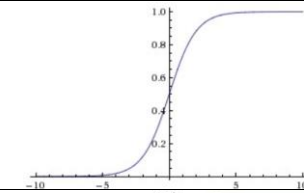
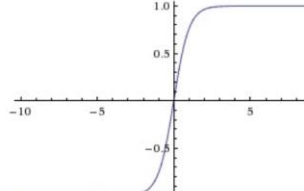
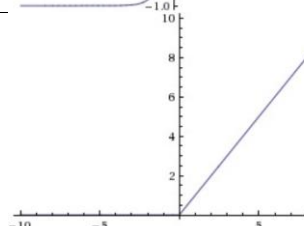
- Deep Neural Network

- Besteht aus:
 - Eingabeschicht (input layer)
 - Ausgabeschicht (output layer)
 - Dazwischen: mehrere Schichten (layer), die von verschiedenen Typen sein können; Abzweigungen sind möglich
 - Kostenfunktion (nach was soll das Netz optimiert werden)



- **Aktivierungsfunktion:**

- Häufig benutzte Aktivierungsfunktionen:

Sigmoid	$f(z) = \sigma(z) = \frac{1}{1+e^{-z}}$ $\frac{d}{dz} f(z) = \sigma(z)(1 - \sigma(z))$	
tanh	$f(z) = \tanh(z)$ $\frac{d}{dz} f(z) = 1 - \tanh^2(z)$	
ReLU	$f(z) = \max(0, z)$ $\frac{d}{dz} f(z) = \begin{cases} 0, & z \leq 0 \\ 1, & z > 0 \end{cases}$ <p>Funktioniert in der Praxis meistens am besten</p>	
Heaviside		
Leaky ReLU		
SeLU		

- **Fully Connected Layer**

- Ein Fully Connected Layer ist eine Hidden Layer eines MLPs
- Alle Inputs sind mit den Neuronen der vorherigen Schicht verbunden
- Parameter: Anzahl von Neuronen (= Dimension des Outputs)
- Alle Neuronen haben dieselbe Aktivierungsfunktion
- Andere Namen: Dense Layer, Linear Layer (ohne Aktivierung)
- $y = \Theta(Wx)$, wobei x die Eingabe und y die Ausgabe des Layers ist

- **Fehlerfunktionen/ Kostenfunktionen**

Mean Squared Error	$L(W) = \frac{1}{2 D } \sum_{d \in D} (t_d - o_d)^2$
Cross- Entropy Loss	<p>Binäre Klassifikation:</p> $L(W) = - \sum_{d \in D} (t_d \log(o_d) + (1 - t_d) \log(1 - o_d))$ <p>Allgemein:</p> $L(W) = - \sum_{d \in D} \sum_{c \in C} (y_{d,c} \log(o_{d,c}))$ $y_{d,c} = \begin{cases} 1, & \text{wenn } d \text{ zu Klasse } c \text{ gehört} \\ 0, & \text{sonst} \end{cases}$ <p>$o_{d,c}$ vorhergesagte Wahrscheinlichkeit, dass d zur Klasse c gehört</p>

- **Softmax Funktion**

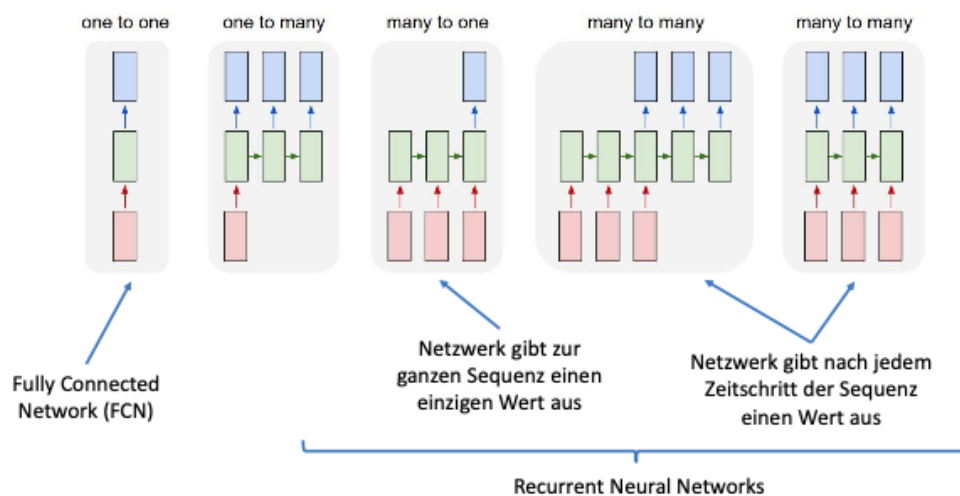
- Oft verwendete Funktion in Deep Learning
- Bildet beliebige Werte x_i zu einer Wahrscheinlichkeitsverteilung p_i ab
→ transformiert die Ausgabe eines Neuronalen Netzes zur Klassenwahrscheinlichkeiten
- $$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^N e^{x_j}} = p_i$$

- **Vermeidung von Overfitting- Dropout**

- Idee: Während des Trainings, „entferne“ einen Teil der Eingänge zufällig mit einer Wsk. p
→ Erzwingt, dass das Netz auf viele Merkmale setzt, nicht nur auf wenige
- „Entfernen“= Input des Neurons auf Null setzen
- Beim Training: Entfernen und skalieren der maskierten Einträge mit $\frac{1}{1-p}$
- Beim Validieren/ Testen: Dropout deaktivieren, um eine deterministische Ausgabe zu erhalten

6.3.2. Recurrent Neural Networks

- Problemdefinition:
 - Gegeben ist eine Sequenz von Eingaben $d = (d_1, d_2, \dots, d_n)$ der Länge n
 - Lerne ein Modell M , das basierend auf der Sequenzeingabe und dem/der letzten Zustand/Zustände die nächste Ausgabe y_{n+1} vorhersagt
 - Trainiere das Problem als Klassifikationsproblem mit den Beispielen: $((d_1, d_2, \dots, d_k), y_{k+1})$
 - Ein Fully Connected (FCN) Layer ist definiert als: $y = f(Wx)$
 - Ein FCN Layer kann nur eine Eingabe verarbeiten, wir haben aber eine Sequenz von Eingaben
 - Mögliche Lösungen:
 - Konkateniere die einzelnen Eingaben d_1, d_2, \dots, d_k zu einem einzigen Vektor $x = \text{concat}(d_1, \dots, d_k)$
 - Führt zu großen Gewichtsmatrizen
 - Unmöglich Sequenzen unterschiedlicher Länge zu bearbeiten
 - Durchschnitt über die Eingabevektoren der Sequenz: $x = \frac{1}{k} \sum_{1 \leq i \leq k} d_i$
 - Eingabevektor bleibt klein
 - Sequenzen unterschiedlicher Länge möglich
 - Alle zeitlichen Informationen gehen verloren



6.3.3. Convolutional Neural Networks

- Typ eines „partially connected“ Feed- Forward Layers
- Ursprung: Computer Vision
- Beispiel: Bildklassifikation (liefern bessere Ergebnisse als klassische Bildklassifikationsverfahren)
- Extrahieren Merkmale über Bereiche im Bild:
 - o Schichten am Anfang extrahieren Merkmale aus den lokalen Nachbarschaften (z.B. Detektion von Kanten)
 - o Schichten am Ende kombinieren die Merkmale um größere Objekte zu erkennen
- CNNs lernen mehrere Filter (z.B. Sharpening (Schärfen), Blurring (Verschmieren), Edge detection (Kantenerkennung)), die auf Bilder angewandt werden können
- Kombinieren diese später dann (durch ein/mehrere FCNs) für die eigentliche Aufgabe