Ludwig Maximilians Universität München
Institut für Informatik
Lehr- und Forschungseinheit für Datenbanksysteme

Skript zur Vorlesung
Managing and Mining Multiplayer Online Games
im Sommersemester 2015

Kapitel 7: Knowledge Discovery und Data Mining in a Nutshell

http://www.dbs.ifi.lmu.de/cms/VO_Managing_Massive_Multiplayer_Online_Games

Kapitelübersicht

- Was ist Knowledge Discovery und Data Mining?
- KDD Prozesse
- Supervised Learning
  - Klassifikation
  - Vorhersage
- Unsupervised Learning
  - Clustering
  - Outlier Detection
- Frequent Pattern Mining
  - Frequent Itemsets

Definition: Knowledge Discovery in Databases

[Fayyad, Piatetsky-Shapiro & Smyth 1996]

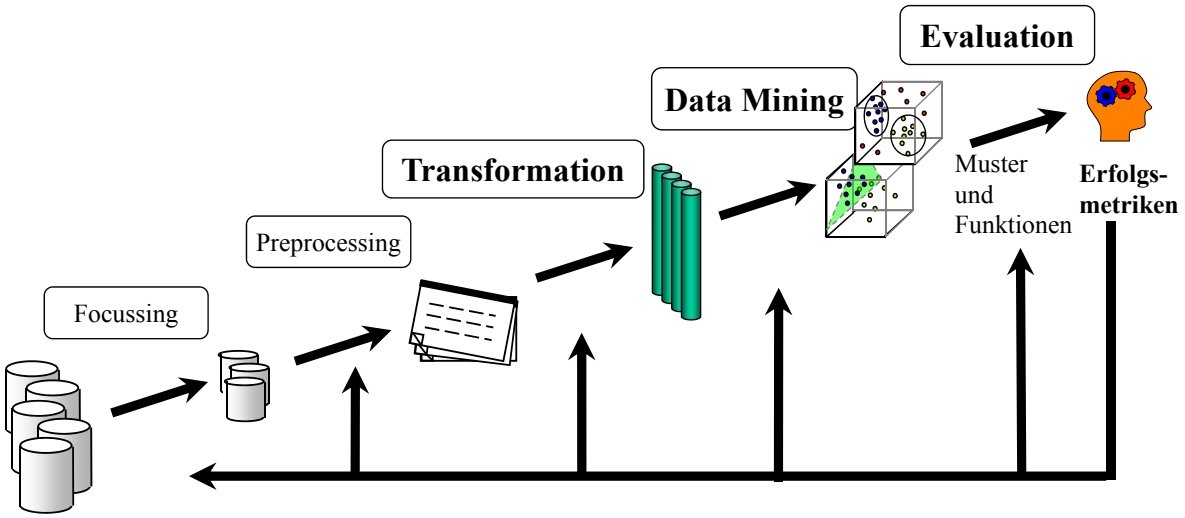
Knowledge Discovery in Databases (*KDD*) ist der Prozess der (semi-)automatischen Extraktion von Wissen aus Datenbanken, das

- *gültig,*
- *bisher unbekannt,*
- und *potentiell nützlich* ist.

Bemerkungen:

- (*semi-*)*automatisch*: im Unterschied zu manueller Analyse. Häufig ist trotzdem Interaktion mit dem Benutzer nötig.
- *gültig*: im statistischen Sinn.
- *bisher unbekannt*: bisher nicht explizit, kein „Allgemeinwissen".
- *potentiell nützlich*: für eine gegebene Anwendung.

Knowledge Discovery Prozesse



- Knowledge Discovery wird in einer Kette aus Teilschritten umgesetzt
- der KDD Prozess wird iterativ immer weiter optimiert (Rückpfeile) bis das Ergebnis akzeptabel ist

  => Es ist notwendig zu wissen, was man eigentlich sucht

Schritte im KDD-Prozess

- **Focussing**: Festlegen eines klaren Ziels und Vorgehens.

  Beispiel: Verwende einen Mitschnitt des TCP-Verkehrs, um ein Vorhersagemodell zu trainieren, das erkennt, ob Spieler von einem Bot gesteuert werden.

- **Preprocessing**: Selektion, Integration und Konsistenzsicherung der zu analysierenden Daten.

  Beispiel: Speichere Mitschnitte des Netzwerkverkehrs von normalen Spielern und Bot-Programmen. Integriere Daten von mehreren Servern. Eliminiere zu kurze und nutzlose (Dauer-AFK) Mitschnitte.

Schritte im KDD Prozess

- **Data Transformation**: Transformiere Daten in eine zur Analyse geeignete Form.

  Beispiel: Bilde für jede beobachtete Minute einen Vektor aus Durchschnittswerten von Paketraten, Paketlängen und Burstiness-Kennzahlen.

- **Data Mining**: Anwendung von effizienten Algorithmen, um statistisch signifikante Muster und Funktionen aus den transformierten Daten abzuleiten.

  Beispiel: Trainiere ein neuronales Netz, das auf Basis von Beispielen für Bots und menschliche Spieler vorhersagt, ob es sich bei einem neuen Mitschnitt um einen Bot handelt oder nicht.

Schritte im KDD Prozess

- **Evaluation**: Qualitätsüberprüfung der Muster und Funktionen aus dem Data Mining.
  - Vergleiche zwischen erwarteten und vorhergesagten Ergebnissen. (Fehlerrate)
  - Manuelle Evaluation durch Experten (Macht das Resultat Sinn?)
  - Evaluation auf Basis von mathematischen Eigenschaften der Muster

  Beispiel: Test auf einer unabhängigen Menge von Testmitschnitten, wie häufig das neuronale Netz einen Bot mit mehr als 50% Konfidenz vorhersagt.

Fazit:

- Fällt der Test nicht zur Zufriedenheit aus, wird der Prozess angepasst.
- Anpassung kann in jedem Schritt erfolgen: mehr Trainingsdaten, andere Algorithmen, andere Parameter, …

Voraussetzungen für die Anwendung

- **Muster und typische Sachverhalte**
  - Muster müssen in irgendeiner Weise vorhanden und erkennbar sein.
  - Daten sollten zu dem gewünschten Ergebnis korreliert sein.
- **Generalisierung und Over-Fitting**
  - Übertragen des Wissens auf neue Objekte benötigt Vergleichbarkeit / Ähnlichkeit zu bereits analysierten Daten.
  - Je weniger Information ein Objekt beschreibt, desto mehr Objekte sind vergleichbar. Je mehr Eigenschaften man betrachtet, desto unterschiedlicher werden die Objekte.
- **Gültigkeit im statistischen Sinn**
  - Wissen erlaubt auch Fehler => keine absoluten Regeln.
  - Nützliches Wissen muss nicht zu 100% korrekt sein, aber mit einem statistisch signifikanten Wert besser sein als zu raten.

Over-Fitting

Überanpassung der Modelle auf die gegebenen Datenobjekte

=> Mangelnde Übertragbarkeit auf andere Datenobjekte

Faktoren die Over-Fitting begünstigen:

- ***Komplexität der Objektbeschreibung***: Je mehr Information vorhanden, desto weniger wahrscheinlich ist es, dass 2 Objekte zueinander ähnlich sind.
- ***Spezifizität von Attributwerten***: Je einzigartiger ein Attributwert ist, desto weniger kann er dazu beitragen viele Elemente bzgl. ihrer Ähnlichkeit zu unterscheiden. (Beispiel: Objekt_ID)
- **Modellkomplexität**: Je komplexer eine Funktion ein Muster aufgebaut ist, desto besser kann es sich exakt auf die gegebenen Objekte einstellen.

Ziel: Modelle, Merkmale und Objektbeschreibungen sollten nicht das Individuum, sondern alle Objekte, die zum gleichen Muster (Klasse, Cluster, …) gehören beschreiben.

Feature-Räume, Distanzen und Ähnlichkeitsmaße

Ähnlichkeit: Objekte, die im Kontext vergleichbar sind.

Beispiel: 2 Spieler, die vom selben Bot gesteuert werden, sollten ähnlichen Netzwerk-Verkehr erzeugen.

- **Feature-Raum**: Sichtweise der Data Mining Algorithmen auf die Objekte. (Merkmale, Struktur, Wertebereiche, …)
- **Ähnlichkeitsmaß**: Berechnet Ähnlichkeit auf Basis des Feature-Raums. (je höher, desto ähnlicher)
- **Distanzmaß**: Berechnet Unähnlichkeit zwischen 2 Objektbeschreibungen. (je höher, desto unähnlicher)

WICHTIG: Feature-Raum und Ähnlichkeitsmaß sind abhängig:

- Änderungen am Feature-Raum ändern das Ergebnis des Maßes.
- Ähnlichkeitsmaß kann nur Teile der Darstellung verwenden oder vorhandene Elemente neu kombinieren.
  (Verändert den verwendeten Datenraum.)

Formale Definition von Distanzfunktionen

Distanzfunktion: Sei ***F*** ein Raum zur Beschreibung eines Objekts.

Dann ist $dist : F \times F \rightarrow \mathrm{IR}^{+}_{0}$ eine totale ***Distanzfunktion*** mit den folgenden Eigenschaften:

- $\forall p,q \in Dom,\ p \neq q : dist(p,q) > 0$ Striktheit
- $\forall o \in Dom : dist(o,o) = 0$ Reflexivität
- $\forall p,q \in Dom : dist(p,q) = dist(q,p)$ Symmetrie

dist heißt ***Metrik***, wenn zusätzlich noch:

- $\forall o,p,q \in Dom : dist(o,p) \leq dist(o,q) + dist(q,p)$ Dreiecksungleichung

gilt.

Vektoren als Objektdarstellung

Feature-Vektoren: Standarddarstellung für die meisten Algorithmen

Grundidee:

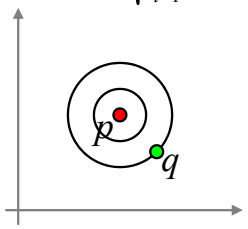
- **Feature**: Merkmal, das das Objekt beschreibt.
  Beispiel: durchschnittliche Anzahl von Paketen pro Sekunde
- **Wertebereiche**:
  - Nominal: Gleichheit und Ungleichheit feststellbar (Beispiel: Name)
  - Ordinal: Werte unterliegen einer Ordnung (Beispiel: Gildenrang)
  - Numerisch: Werte haben einen quantifizierbaren Unterschied (Level (diskret), Paketrate (metrisch), …)
- **Feature-Vektor**: Gesamtheit aller beschreibenden Merkmale
  Beispiel:(Name, Gildenrang, Level, ∅Paketrate, ∅Paketgröße)
- Bei rein numerischen Daten existiert eine Vielzahl von algebraischen Funktionen und Gesetzmäßigkeiten, die zur Analyse einsetzbar sind.

L_p-Metriken in Vektorräumen

Für $p,q \in IR^d$:

Euklidische Norm (L_2):

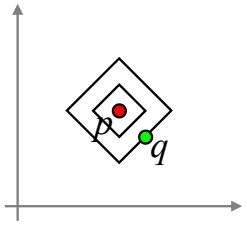
$$d_2(p,q) = \sqrt{\sum_{i=1}^{d} (p_i - q_i)^2}$$



Natürlichstes Distanzmaß

Manhattan-Norm (L_1):

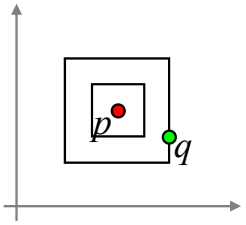
$$d_1(p,q) = \sum_{i=1}^{d} |p_i - q_i|$$



Die Unähnlichkeiten der einzelnen Merkmale werden direkt addiert

Maximums-Norm (L_∞):

$$d_\infty(p,q) = \max_{1 \le i \le d} |p_i - q_i|$$



Die Unähnlichkeit des am wenigsten ähnlichen Merkmals zählt

Allgemeines L_p-Distanzmaß: $d_p(p,q) = \left(\sum_{i=1}^{d} |p_i - q_i|^p \right)^{\frac{1}{p}}$

Normen, Ähnlichkeiten und Kernel

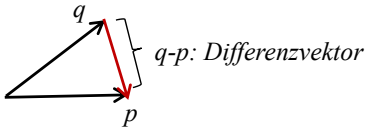
- *Euklidische Distanz*: Länge des Differenzvektors
- *Länge eines Vektors*: Norm des Vektors
- *Norm*: Skalarprodukt mit sich selbst
- Eigenschaften eines Skalarprodukts:

  $\langle \cdot,\cdot \rangle : F \times F \to IR,$

  $\langle c \cdot x, y \rangle = \langle x, y \cdot c \rangle = c \langle x, y \rangle$

  $\langle x, y \rangle = \langle y, x \rangle$

  $\langle x + y, z \rangle = \langle x, z \rangle + \langle y, z \rangle$



$\|q - p\|$

$$\|\vec{x}\| = \sqrt{\langle \vec{x}, \vec{x} \rangle} = \sqrt{\sum_{i=1}^{d} x_i \cdot x_i}$$

- Zusammenhang zwischen Skalarprodukten, Normen und Metriken:

  $\|q - p\| = \langle q - p, q - p \rangle^{\frac{1}{2}} = \left(\langle q, q \rangle + \langle p, p \rangle - 2 \langle q, p \rangle \right)^{\frac{1}{2}}$

  (Skalarprodukte implizieren Normen und Normen implizieren Metriken)
- Skalarprodukte werden häufig als Ähnlichkeitsmaße verwendet.

  (In diesem Zusammenhang nennt man sie ***Kernel***-Funktionen.)

Supervised Learning

Idee: Lernen aus Beispielobjekten zur Optimierung einer Vorhersagefunktion.

Gegeben:

- Zielvariable *C*
  (*Klassifikation*: Menge aus nominalen Werten, *Regression*: numerische Werte)
- Objekte: $DB \in F \times C$: *Object* $o=(o.v,o.c) \in DB$
- Trainingsmenge: T $\subseteq$ DB bei der *o* vollständig bekannt ist.

Gesucht: Funktion $f: F \rightarrow C$, die Objektdarstellungen auf Werte der Zielvariable abbilden und dabei möglichst wenige Fehler macht.

Fehlerfunktion: Quantifiziert die Güte des Modells auf T.

Quadratischer Fehler: $L^2(f,T) = \sum_{o \in T} \left(o.c - f(o.v)\right)^2$

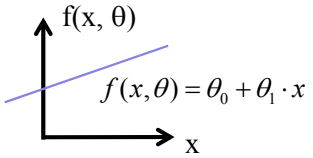
Absoluter Fehler: $L_{abs}(f,T) = \sum_{o \in T} \left(o.c = f(o.v)\,?1:0\right)$

Training von Supervised Methoden

- Art der Funktion f ist i.d.R. bereits vorgegeben, z.B. lineares Modell
- Anpassung von f auf die Trainingsdaten T erfolgt mit Hilfe der Parameter θ

Beispiel: f ist eine lineare Funktion:

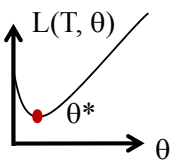
$$f(x,\theta) = \theta_0 + \sum_{i=1}^{d} \theta_i \cdot x_i$$



Lösung im 2D-Raum

Training: Minimiere die Fehlerfunktion

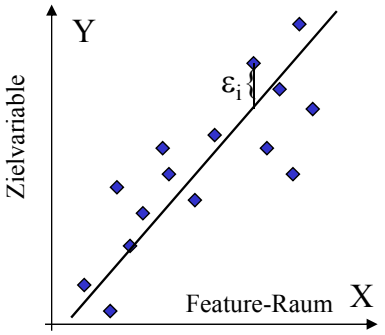
- Fehler Funktion *L* berechnet den Fehler den *f* für *T* macht
- Wähle die Parameter θ*, die *L* minimieren
- ***Vorgehen***: Differenzieren der Fehlerfunktion nach θ und Suche nach dem Minimum θ*.
  => Fehlerfunktion sollte konvex sein
  (nur ein Extremum und das ist ein Minimum)
  => allgemeine Fehlerfunktionen können in lokalen Minima stagnieren



Beispiel: Lineare Regression

Gegeben: Trainingsmenge $T \in IR^2$

Model: $f(x,\theta) = \theta_0 + \theta_1 \cdot x$



Fehlerfunktion:

$$L^2(f,T) = \sum_{(x,y)\in T} (y - f(x,\theta))^2 = \sum_{(x,y)\in T} \left(y^2 + f(x,\theta)^2 - 2y \cdot f(x,\theta)\right)$$

$$= \sum_{(x,y)\in T} \left(y^2 + (\theta_1 x)^2 + \theta_0^2 + 2\theta_0\theta_1 x - 2y\theta_0 - 2yx\theta_1\right)$$

Ableitung nach θ_0: $\frac{\partial L^2}{\partial \theta_0} = \sum_{(x,y)\in T} (2\theta_0 - 2y + 2\theta_1 x) = 2\left(\theta_0 |T| - \sum_{(x,y)\in T} y + \theta_1 \sum_{(x,y)\in T} x\right)$

$$\frac{\partial L^2}{\partial \theta_0} = 0: \quad \theta_0 = \frac{\sum_{(x,y)\in T} y - \theta_1 \sum_{(x,y)\in T} x}{|T|} = E(y) - \theta_1 E(x)$$

Ableitung nach θ_1:

$$\frac{\partial L^2}{\partial \theta_1} = \sum_{(x,y)\in T} (2\theta_1 x^2 - 2yx + 2\theta_0 x) = 2\left(\theta_1 \sum_{(x,y)\in T} x^2 + E(y) \sum_{(x,y)\in T} x - \theta_1 E(x) \sum_{(x,y)\in T} x - \sum_{(x,y)\in T} yx\right)$$

$$\frac{\partial L^2}{\partial \theta_1} = 0: \quad \theta_1 = \frac{E(y) \sum_{(x,y)\in T} x - \sum_{(x,y)\in T} yx}{\sum_{(x,y)\in T} x^2 - E(x) \sum_{(x,y)\in T} x} = \frac{Cov(x,y)}{Var(x)}$$

Weitere Anmerkungen zum Supervised Learning

- **Regularisierung**: Häufig ist die Wahl der Parameter so frei, dass Overfiting möglich wird (z.B. verkleinern aller Parameter verringert Zielfunktion).
  => Integriere Regularisierungsterm in die Zielfunktion, die Freiheitsgrade einschränkt.

  Beispiel: lineare Ridge-Regression

  Regularisierungsterm

$$L^2(f,T) = (1 - \alpha \cdot) \sum_{o\in T} \left(o.c - \theta_0 + \sum_{i=1}^{d} \theta_i \cdot o.v_i\right)^2 + \alpha \cdot \|\theta\|^2$$

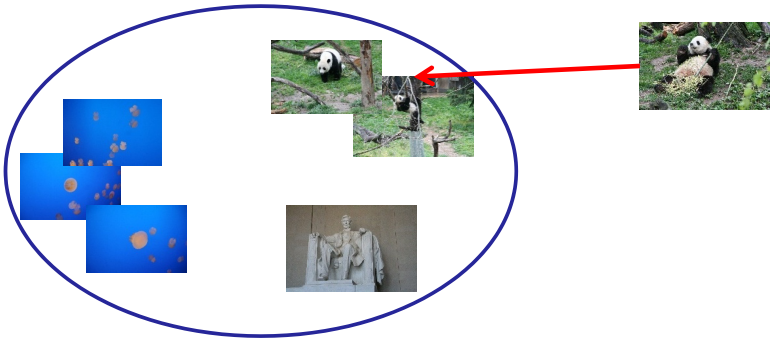
- Weitere Problemformulierungen basieren häufig auf Optimierungsproblemen mit Nebenbedingungen und sind daher wesentlich aufwendiger (quadratische Programme, semidefinite Programme, …)
- Zielfunktionen müssen nicht unbedingt konvex sein (Neuronale Netze)
  => Optimierung findet evtl. nur lokale Extrema
- Es gibt noch wesentlich mehr Techniken, die Zielvariablen vorhersagen, die nicht direkt eine Fehlerfunktion minimieren.

Instanzbasiertes Lernen

Grundidee: Suche die ähnlichsten Objekte in der Trainingsmenge und verwende deren Zielwerte für die Vorhersage.

Komponenten:

- Bestimmung der Entscheidungsmenge:
  - abhängig vom Ähnlichkeits-/Distanzmaß (k-nächste Nachbarn in *T*)
  - Größe *k* wird vorgegeben und regelt die Generalisierung der Vorhersage
- Bestimmung der Vorhersage
  - Mehrheitsklasse in der Entscheidungsmenge
  - Gewichtung nach Distanz *(z.B. quadratisch inverse Gew.: $1/d(q,x)^2$)*



Bayes‘sches Lernen

Grundidee: Jeder Beobachtung liegt ein bestimmter statistischer Prozess zugrunde (Verursacher). Kennt man alle Prozesse, kann man die Wahrscheinlichkeit dafür berechnen, dass unter der gegebenen Beobachtung ein bestimmter Prozess die Ursache ist.

***Beispiel**: Bot-Detection*

Gegeben: Modell *A*: Mensch spielt

Modell *B*: Bot spielt

Beobachtung: *v* (Vektor über Netzwerkmitschnitt)

Annahme: *v* wurde entweder von *A* oder *B* verursacht.

Aufgabe: Berechne die Wahrscheinlichkeit mit der *v* von *B* verursacht wurde.

- **ACHTUNG**: Lösung ist **NICHT** $P(v|B)$, die Wahrscheinlichkeit, dass *B* genau die Beobachtung *v* hervorruft.
- Selbst wenn $P(v|B) = 0.1$ ist, heißt das nicht, dass Modell B *v* wahrscheinlicher verursacht hat als *A* mit $P(v|A) = 0.01$.
- Lösung ist $P(B|v)$, die Wahrscheinlichkeit das *B* spielt unter dem Wissen das der Spieler die Beobachtung *v* erzeugt hat.

Der Satz von Bayes

Lösung: Gesucht ist *P(A|v)* bzw. *P(B|v)*
(Wahrscheinlichkeit, dass *B* spielt unter der Beobachtung *v*)

- Aus der Annahme folgt, dass $p(v) = p(A)\cdot p(v|A) + p(B)\cdot p(v|B)$, wobei *P(B)* bzw. *P(A)* beschreibt wie wahrscheinlich *B* bzw. *A* wirkliche spielen. (a priori Wahrscheinlichkeit)

  D.h., selbst wenn *v* eher durch *B* (Bot) verursacht wird, aber *B* nur sehr selten eingeschaltet wird, kann das dazu führen, dass *P(A|v)>P(B|v)* gilt.

Allgemein:

- Satz von Bayes besagt: $P(B \mid v) = \frac{P(B)\cdot P(v \mid B)}{P(v)}$
- Für alle Prozesse C und die Beobachtung *v* gilt: $\sum_{c \in C} P(c \mid v) = 1$

  (irgendein Prozess muss die Beobachtung verursacht haben)
- Daher ist c* mit $c^* = \arg\max_{c \in C}\left(P(c \mid v)\right)$ der wahrscheinlichste Verursacher

Training von Bayes Klassifikatoren

- A priori Wahrscheinlichkeiten *P(c)* werden i.d.R. durch relative Häufigkeiten der Klassen in der Trainingsmengen *T* beschrieben.

  (17 von 100 Mitschnitten sind Bots: *P(B)* =17%)
- Der Bestimmung von *p(v|c)* liegen Verteilungsmodelle zugrunde, wobei sich die Modelle für die $c \in C$ meist nur durch Parameter unterscheiden.
- Training über Berechnung der relativen Häufigkeiten aus den Trainingsdaten die zu Modell c gehören.

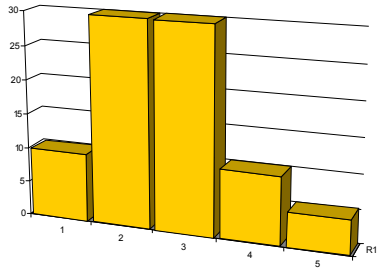
Beispiel: Betrachte 2 Würfel

- Mögliche Ereignisse: {1, 2, 3, 4, 5, 6}
- Würfel W1 ist gleichverteilt: 1/6 für 1 bis 6
- Würfel W2 Verteilung: 1: 1/12, 2: 1/12, 3: 1/6, 4: 1/6, 5: 1/6, 6: 1/3
- *p(v=1|W1) = 1/6, p(v=6|W2)=1/3*
- Sei *P(W1)= 0.2* und *P(W2)=0.8:*

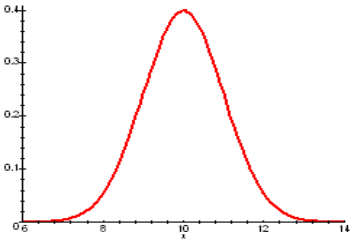
$$P(W2 \mid 5) = \frac{0{,}8\cdot 0{,}1\overline{6}}{0{,}2\cdot 0{,}1\overline{6} + 0{,}8\cdot 0{,}1\overline{6}} = 0.8; \quad P(W2 \mid 6) = \frac{0{,}8\cdot 0{,}\overline{3}}{0{,}2\cdot 0{,}1\overline{6} + 0{,}8\cdot 0{,}\overline{3}} = \frac{8}{9}$$

Arten von univariaten Verteilungen

- diskrete Zufallsräume:
  - endliche Anzahl von Ereignissen
  - getrennte Abschätzung für alle möglichen Grundereignisse



- reelle Zufallsräume:
  - unendlich viel Ereignisse
    (jedes Ereignis ist unendlich unwahrscheinlich: 1/∞)
  - Abschätzung über Wahrscheinlichkeitsdichtefunktionen (z.B. Normalverteilung)
  - Training über Parameter der Dichtefunktion (z.B. Erwartungswert und Varianz)
  - Wahrscheinlichkeiten werden aus Dichtefunktionen mittels ***Integration*** oder dem ***Satz von Bayes*** abgeleitet.



Statistische Modelle

Bei kombinierter Abschätzung von d Beobachtungen muss $p(v_1, \ldots, v_d|c)$ berechnet werden.

Problem: Sind $v_1, \ldots, v_d$ abhängig voneinander?

- ***naive Annahme***: Alle Beobachtungen sind unabhängig voneinander
  => naive Bayes

  Vorteil: Berechnung wird einfach, da gilt: $P(v_1, \ldots, v_d \mid c) = \prod_{i=1}^{d} P(v_i \mid c)$

  Nachteil: Zusammenhängende Beobachtungen beeinflussen das Ergebnis stärker als gewollt.
- ***vollständige Abhängigkeit***: Alle Beobachtungen werden als eine kombinierte Beobachtung abgeschätzt.

  Vorteil: Berechnung stimmt selbst bei unabhängigen Parametern.

  Nachteil: Anzahl der Grundereignisse steigt exponentiell mit den Beobachtungen => Stichproben meist viel zu klein.

Statistische Modelle

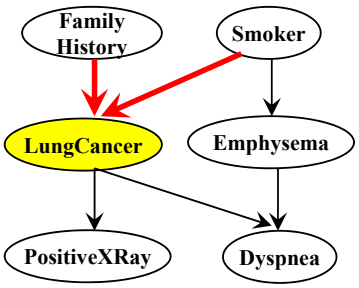
Fortgeschrittene Lösungen:

=> Betrachte nur bestimmte Abhängigkeiten

Beispiel Bayes Netzwerke

Ordnet die möglichen Einflussgrößen in einem gerichteten Netzwerk.

- Abschätzen der Verteilungen eines abhängigen Parameters erfolgt als bedingte Wahrscheinlichkeit, die von den Werten der Vorgänger im Netz abhängt.
- Netzwerktopologie wird durch Domainen-Wissen vorgegeben oder durch Heuristiken automatisch bestimmt.



	FH,S	FH, ¬S	¬FH,S	¬FH, ¬S
LC	0.8	0.5	0.7	0.1
~LC	0.2	0.5	0.3	0.9

Bewertung im Supervised Learning

- Das Optimum der Zielfunktion ist nicht aussagekräftig, da es auf den Trainingsdaten over-fitten kann.

  ⇒ Es ist entscheidend, wie gut ein Modell auf unbekannten Daten funktioniert. (Generalisierung)

- Ein vernünftiger Test erfordert die Verwendung einer unabhängigen Testmenge, die nicht im Training verwendet wurde. (**Train and Test** Methode)

- **Problem**: Datenobjekte mit vorhandener Zielvariable sind in der Regel Mangelware. (manueller Aufwand bei der Zuordnung)

Testschemata für das Supervised Learning

Ziele:

- Teste und trainiere auf möglichst vielen Instanzen.
- Schließe Over-Fitting aus. (Trainings und Testmenge sind disjunkt)

Lösungsansätze:

- **Leave-One-Out**:
  - Für *n* Datenobjekte werden *n* Tests durchgeführt.
  - In jedem Durchgang wird genau 1 bisher ungetestetes Objekt in die Testmenge verschoben und auf allen anderen trainiert.
  - Jedes Modell wird nur auf das im Training ausgelassene Objekt angewendet.
  - Aufwand für das Training ist maximal.
  - Ergebnisse sind wiederholbar.
  - Nur für kleine Datenmengen oder instanzbasierte Verfahren gut geeignet.

Stratified *k*-fold Cross Validation

- Ähnlich wie Leave-One-Out, aber anstatt eines werden n/k Datenobjekte (fold) aus der Trainingsmenge in die Testmenge verschoben und getestet.

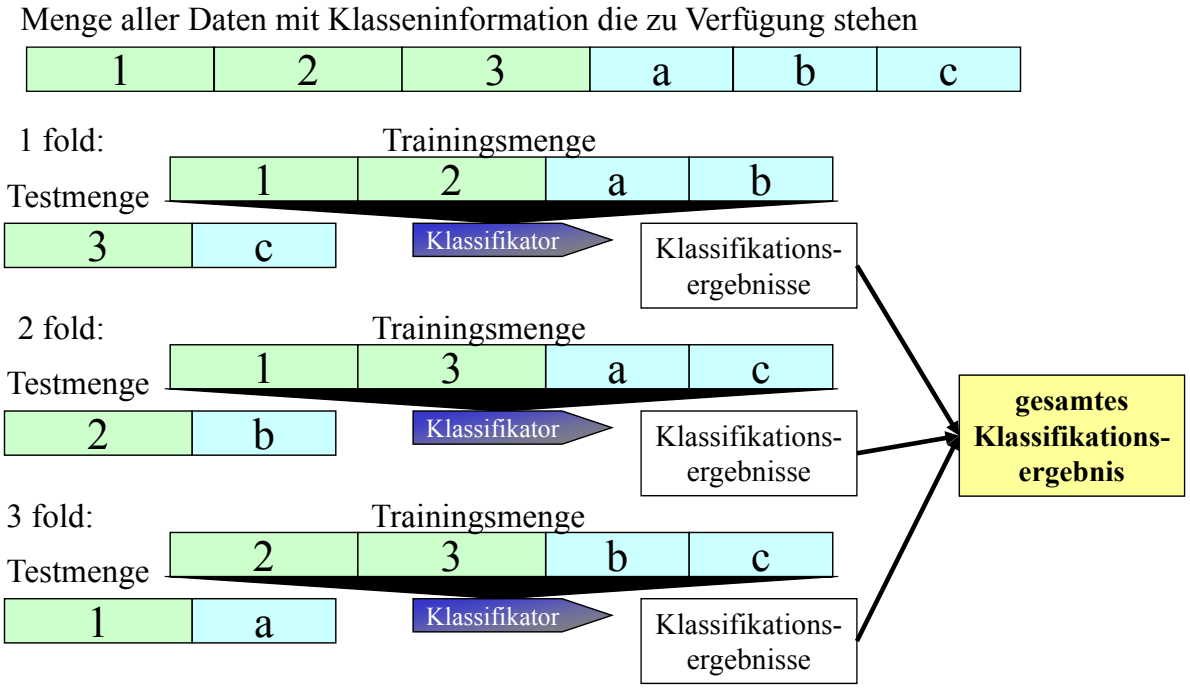
- **Stratifizierung**: In jeder Teilmenge (fold) sollte möglichst die gleiche Verteilung über die Klassen gelten wie in der gesamten Datenmenge. (besserer Erhalt der a priori Wahrscheinlichkeit)

- Die Anzahl der Teilmengen (folds) *k* wird je nach Größe der Datenmenge angepasst :
  - je mehr Objekte, desto länger dauert das Training
  - je weniger Objekte, desto weniger Objekte werden zum Training verwendet

- *k*-fold Cross Validation hängt von der Bildung der Teilmengen ab.
  => Ergebnis kann je nach Reihenfolge variieren.
  => *k*-fold Cross Validation wird häufig *l* mal durchgeführt und das Ergebnis gemittelt.

Beispiel: Stratified 3-fold Cross Validation

grüne Kästchen: Klasse 1 (Folds:1, 2, 3)
blaue Kästchen: Klasse 2 (Folds: a, b, c)



Bewertung von Klassifikatoren

Ergebnis des Tests : Konfusionsmatrix (confusion matrix)

klassifiziert als ...

tatsächliche Klasse ...	Klasse 1	Klasse 2	Klasse 3	Klasse 4	Klasse 5
Klasse 1	35	1	1	1	4
Klasse 2	0	31	1	1	5
Klasse 3	3	1	50	1	2
Klasse 4	1	0	1	10	2
Klasse 5	3	1	9	15	13

korrekt klassifizierte Objekte

Aus der Konfusionsmatrix lassen sich u.a. folgende Kennzahlen berechnen:
Genauigkeit, Klassifikationsfehler, Precision und Recall.

Gütemaße für Klassifikatoren

- Sei *f* ein Klassifikator, $TR \subseteq DB$ die Trainingsmenge, $TE \subseteq DB$ die Testmenge
- *o.c* bezeichnet die tatsächliche Klasse eines Objekts *o*
- *f(o)* die von *f* vorhergesagte Klasse
- *Klassifikationsgenauigkeit* (classification accuracy) von *K* auf *TE*:

$$G_{TE}(f) = \frac{|\{o \in TE | f(o) = o.c\}|}{|TE|}$$

- *Tatsächlicher Klassifikationsfehler* (true classification error)

$$F_{TE}(f) = \frac{|\{o \in TE | f(o) \neq o.c\}|}{|TE|}$$

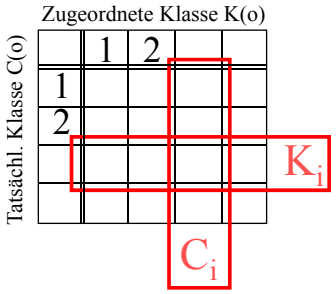
- *Beobachteter Klassifikationsfehler* (apparent classification error)

$$F_{TR}(f) = \frac{|\{o \in TR | f(o) \neq o.c\}|}{|TR|}$$

Bewertung von Klassifikatoren

- *Recall:*
  Anteil der zur Klasse *i* **gehörenden** Testobjekte, die richtig erkannt wurden.
  Sei $C_i = \{o \in TE \mid o.c = i\}$, dann ist

$$Recall_{TE}(f,i) = \frac{|\{o \in C_i | f(o) = o.c\}|}{|C_i|}$$



- *Precision*:
  Anteil der zu einer Klasse *i* **zugeordneten** Testobjekte, die richtig erkannt wurden.
  Sei $K_i = \{o \in TE \mid f(o) = i\}$, dann ist

$$Precision_{TE}(f,i) = \frac{|\{o \in K_i | f(o) = o.c\}|}{|K_i|}$$

Unsupervised Learning

Problemstellung: Es sind nur Datenobjekte gegeben, aber keine Beispiele die Werte der Zielvariable mit Werten aus der Darstellung im Feature-Raum verknüpfen.

Aufgaben:

- Finde Gruppen von ähnlichen Objekten. (Clustering)
- Finde Objekte, die ungewöhnlich sind. (Outlier Detection)
- Finde Teile von Objektbeschreibungen, die häufig vorkommen. (Pattern Mining)

Vorteile:

- Ergebnis ist generischer, da es weniger Angaben über die Zielvariablen gibt.
- Keine manuelle Bewertung von Trainingsbeispielen notwendig. (vorher)

Nachteile:

- Es ist schwierig, die Ergebnisse in Relation zum Vorhersageziel zu setzen.
- Höhere Flexibilität hat häufig auch höhere Komplexität zur Folge.
- Manuelle Bewertung wird nach dem Data Mining notwendig, um das Ergebnis zu evaluieren.

Beispielanwendungen

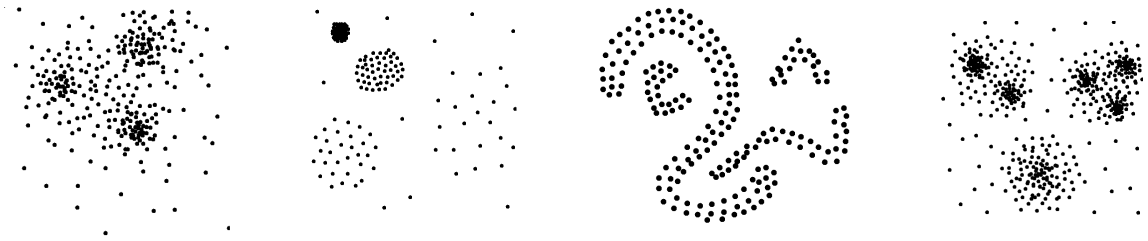
- **Clustering**: Welche Taktiken werden beim Kampf gegen einen Boss-Gegner eingesetzt?
  (Darstellung der Taktik und Clustering der Ergebnisse)

- **Outlier Detection**: Welche Spieler verhalten sich verdächtig?
  (Beschreibung der Spieler durch Online-Zeiten und Aktivitätsverteilung => feststellen der Abweichung von „normalen Spielern“)

- **Pattern Mining**: Finde Standardrotationen bei einem RPG
  (Abbilden des Spielerverhaltens durch die Reihenfolge der eingesetzten Fähigkeiten => häufige Muster entsprechen einem typischen Verhalten)

Clustering Verfahren

- Identifikation einer endlichen Menge von Kategorien, Klassen oder Gruppen (*Cluster*) in den Daten.
- *Ähnliche* Objekte sollen im *gleichen* Cluster sein, *unähnliche* Objekte sollen in *unterschiedlichen* Clustern sein.
- Clustering-Modelle sollen sowohl Klassen (Cluster) finden, als auch Methoden liefern, Objekte den Klassen (Clustern) zu-zuweisen.



Clustering

Gegeben:

- Eine Objektmenge: $DB \subseteq F$
- Eine diskrete Zielvariable $C \subseteq IN_0$ (Menge der Clusterindizes)
- ggf. ist $|C|$ bekannt (Anzahl der Cluster)

Gesucht: Funktion $f: F \rightarrow C$, die Objektdarstellungen auf Werte der Zielvariable abbilden und dabei möglichst „*gute Cluster*“ findet.

Güte eines Clusters:

- Abhängig vom Clustermodell:
  - Wie wird festgestellt, dass Objekt o zu Cluster c gehört?
  - Wie kann man entscheiden, ob 2 Objekte im gleichen Cluster liegen?
- Optimiert werden sollen:
  - Kompaktheit des Clusters
  - Trennung der Cluster

Partitionierendes Clustering (1)

Idee:

- Es gibt k Cluster und jeder Cluster *c* hat ein repräsentatives Objekt o_c
- Zuordnung von Objekten *o* zu *c* über den Abstand *dist*(o_c,*o*):
  $cluster(o) = \arg\min_{c \in C}(dist(o_c, o))$
- Kompaktheit eines Clusters über:
  - durchschnittlichen Abstand der zugeordneten Elemente:
    $compact(c) = \sum_{o \in \{o \in DB \mid cluster(o)=c\}} dist(o_c, o)$
  - durchschnittlichen quadratischen Abstand der zugeordneten Elemente:
    $sqrComp(c) = \sum_{o \in \{o \in DB \mid cluster(o)=c\}} dist(o_c, o)^2$
- Güte des Clusterings: $TD(C) = \sum_{c \in C} compact(c)$

  oder $TD^2(C) = \sum_{c \in C} sqrComp(c)$

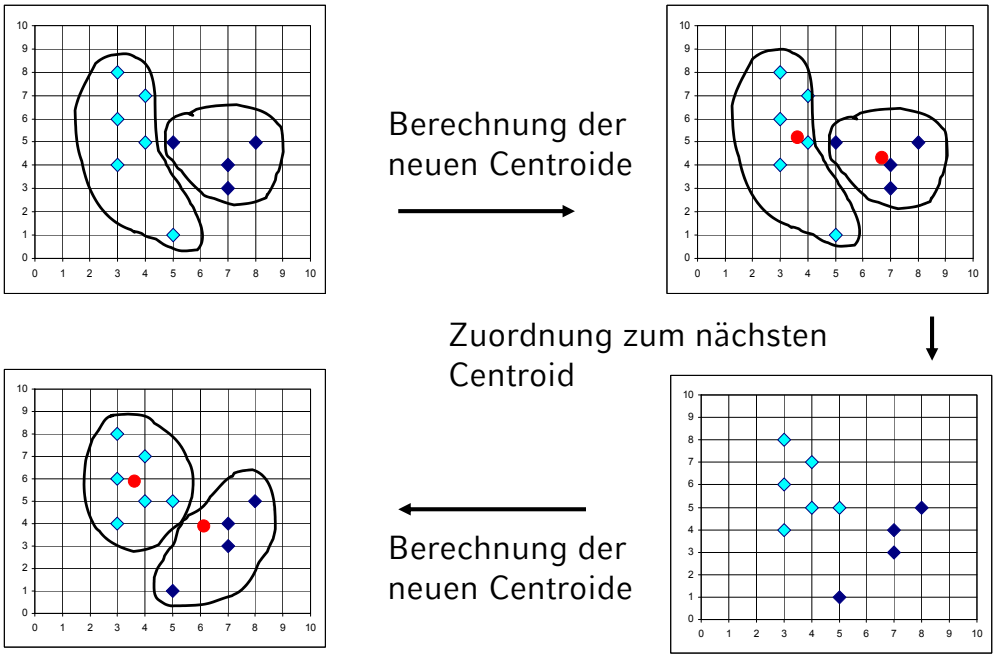
Partitionierendes Clustering (2)

- typische Repräsentanten:
  - Centroid: $centroid(c) = \frac{1}{|\{o \in DB \mid cluster(o) = c\}|} \sum_{o \in \{o \in DB \mid cluster(o)=c\}} o$
  - Medoid: $medoid(c) = \arg\min_{o \in \{o \in DB \mid cluster(o)=c\}} \left(\sum_{p \in \{p \in DB \mid cluster(o)=c\}} dist(o, p) \right)$

Minimieren von TD oder TD^2:

- *TD* und TD^2 sind nicht konvex und haben lokale Minima
- *TD* und TD^2 sind nicht stetig (Sprungstellen bei Clusterwechsel möglich)
- Verfahren mit Greedy-Suche die TD/ TD^2 lokal minimieren
  1. Schritt: Gegeben ist cluster(o) für alle o ∈ DB
     => bilde die Menge der Clusterrepräsentaten $\{o_{c1}, ..., o_{cn}\}$
  2. Schritt: Gegeben ist *Centroids* = $\{o_{c1}, ..., o_{cn}\}$
     => ordne alle Objekte *o* dem nächsten Element von *Centroids* zu
  3. Abbruch, falls sich TD/ TD^2 nicht mehr ändert

Beispiel für partitionierendes Clustering



Algorithmus

```
ClusteringDurchVarianzMinimierung(Objektmenge DB, Integer k)
    Erzeuge eine „initiale" Zerlegung
      der Objektmenge DB in k Cluster;
    Berechne die Menge C'={C_1, ..., C_k} der
      Centroide für die k Cluster;
    C = {};
    TD2 = sqrTD(C',DB);
    repeat
        TD2old = TD2;
        C = C';
        Bilde k Cluster durch Zuordnung jedes Punktes zum
          nächstliegenden Centroid aus C;
        Berechne die Menge C'={C'_1, ..., C'_k} der Centroide für
          die neu bestimmten Cluster;
        TD2 = sqrTD(C',DB);
    until TD2 == TD2old;
    return C;
```

partitionierendes Clustering

Varianten:

- *k*-Means: in jedem Schritt wird nur ein Objekt ggf. neu zugewiesen.
  Der alte und der neue Cluster-Centroid wird danach gleich geändert.
- Expectation Maximation Clustering (EM)
  Statistische Variante, die Cluster durch Verteilungfunktionen beschreibt.
- k-Medoid Clusterings:
  - Cluster-Repräsentanten sind Medoide
  - Anpassung der Cluster durch vertauschen von Medoiden und Nicht-Medoid Objekten.

Eigenschaften:

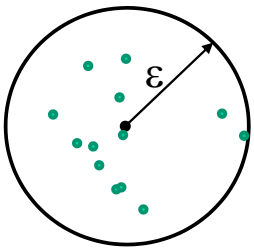
- Algorithmus ist stark von der initialen Partitionierung abhängig.
- Centroid-basierte Verfahren sind sehr schnell $O(i \cdot n \cdot k)$. (#Iterationen i)
- Medoid Verfahren sind relativ langsam $O(i \cdot n^2 \cdot k)$ (#Iterationen i)

Dichtebasiertes Clustering

Idee: Cluster sind dichte Regionen im Feature-Raum F.

Dichte:

$$\frac{\text{Anzahl Objekte}}{\text{Volumen}}$$



Hier:

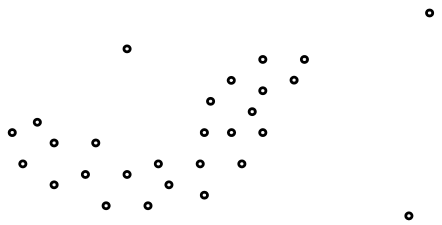
- ***Volumen***: ε-Umgebung bzgl. Distanzmaß *dist(x,y)* um Objekt o
- ***dichte Region:*** es liegen MinPts Objekte in der ε-Umgebung von o
  => o wird Kernpunkt genannt
- „zuhängende Kernpunkte“ bilden ***Cluster***
- Punkte, die in keinem Cluster liegen nennt man Rauschen (***Noise***)

Dichtebasiertes Clustering

Intuition

Parameter $\varepsilon \in IR$ und $MinPts \in IN$ spezifizieren den Dichtegrenzwert

ε $MinPts = 4$



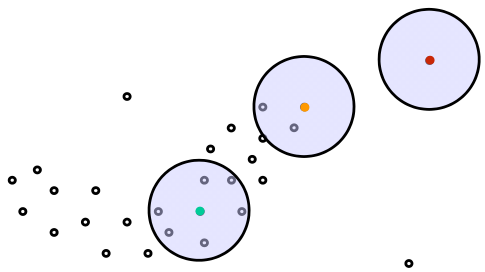
Dichtebasiertes Clustering

Intuition

Parameter $\varepsilon \in IR$ und $MinPts \in IN$ spezifizieren den Dichtegrenzwert

ε $MinPts = 4$

- *Kernpunkt*



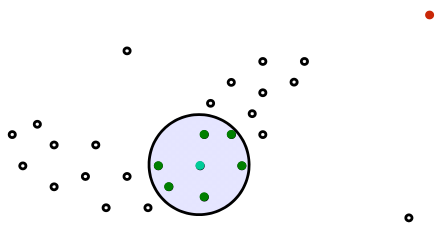
Dichtebasiertes Clustering

Intuition

Parameter $\varepsilon \in IR$ und $MinPts \in IN$ spezifizieren den Dichtegrenzwert

ε $MinPts = 4$

- *Kernpunkt*
- *Direkte Dichte-Erreichbarkeit*



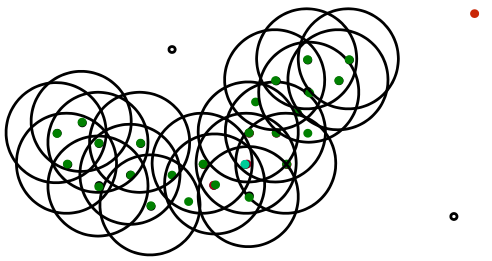
Dichtebasiertes Clustering

Intuition

Parameter $\varepsilon \in IR$ und $MinPts \in IN$ spezifizieren den Dichtegrenzwert

ε $MinPts = 4$

- *Kernpunkt*
- *Direkte Dichte-Erreichbarkeit*
- *Dichte-Erreichbarkeit*



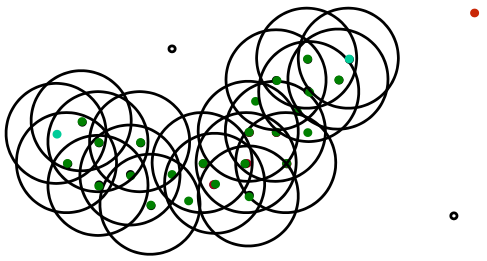
Dichtebasiertes Clustering

Intuition

Parameter $\varepsilon \in IR$ und $MinPts \in IN$ spezifizieren den Dichtegrenzwert

ε $MinPts = 4$

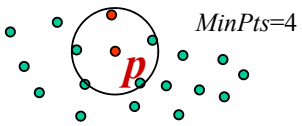
- *Kernpunkt*
- *Direkte Dichte-Erreichbarkeit*
- *Dichte-Erreichbarkeit*
- *Dichte-Verbundenheit*



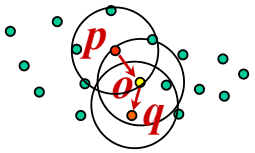
Dichtebasiertes Clustering

Formalisierung [Ester, Kriegel, Sander & Xu 1996]

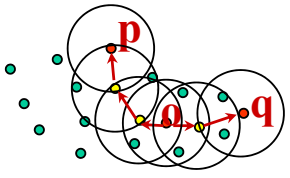
- Ein Objekt $p \in DB$ heißt *Kernobjekt*, wenn gilt:
  $|RQ(p,\varepsilon)| \geq MinPts$
  $RQ(p,\varepsilon) = \{o \in DB \mid dist(p,o) <= \varepsilon\}$



- Ein Objekt $p \in DB$ ist *direkt dichte-erreichbar* von $q \in DB$ bzgl. ε und *MinPts,* wenn gilt:
  $p \in RQ(q,\varepsilon)$ und q ist ein Kernobjekt in *DB*.



- Ein Objekt p ist *dichte-erreichbar* von q,
  wenn es eine Kette von direkt erreichbaren Objekten von q nach p gibt.
- Zwei Objekte p und q sind *dichte-verbunden*,
  wenn sie beide von einem dritten Objekt o aus dichte-erreichbar sind.



Formalisierung

Ein (*dichtebasierter*) *Cluster* C bzgl. ε und *MinPts* ist eine nicht-leere Teilmenge von *DB,* für die die folgenden Bedingungen erfüllt sind:

Maximalität: $\forall p,q \in DB$: wenn $p \in C$ und q dichte-erreichbar von p ist, dann ist auch $q \in C$.

Verbundenheit: $\forall p,q \in C$: p ist dichte-verbunden mit q.

Formalisierung

- Definition Clustering
  Ein *dichtebasiertes Clustering* *CL* der Menge *DB* bzgl. ε und *MinPts* ist eine „vollständige“ Menge von dichtebasierten Clustern bzgl. ε und *MinPts* in *DB*.

- Definition Noise
  Die Menge $Noise_{CL}$ („*Rauschen*“) ist definiert als die Menge aller Objekte aus *DB*, die nicht zu einem der dichtebasierten Cluster $C \in CL$ gehören.

- Grundlegende Eigenschaft
  Sei C ein dichtebasierter Cluster und sei $p \in C$ ein Kernobjekt.
  Dann gilt:
  $C = \{o \in DB \mid o$ dichte-erreichbar von p bzgl. ε und *MinPts*$\}$.

  => ermöglicht effiziente Suche (WARUM?)

Dichtebasiertes Clustering

Algorithmus DBSCAN

```
DBSCAN(Punktmenge DB, Real ε, Integer MinPts)
  // Zu Beginn sind alle Objekte unklassifiziert,
  // o.ClId = UNKLASSIFIZIERT für alle o ∈ DB

   ClusterId := nextId(NOISE);
   for i from 1 to |DB| do
      Objekt := DB.get(i);
      if Objekt.ClId = UNKLASSIFIZIERT then
         if ExpandiereCluster(DB, Objekt, ClusterId, ε, MinPts)
         then ClusterId:=nextId(ClusterId);
```


Dichtebasiertes Clustering

```
ExpandiereCluster(DB, StartObjekt, ClusterId, ε, MinPts): Boolean
seeds:= RQ(StartObjekt, ε);
if |seeds| < MinPts then // StartObjekt ist kein Kernobjekt
  StartObjekt.ClId := NOISE;
  return false;
// sonst: StartObjekt ist ein Kernobjekt
forall o ∈ seeds do o.ClId := ClusterId;
entferne StartObjekt aus seeds;
while seeds ≠ Empty do
  wähle ein Objekt o aus der Menge seeds;
  Nachbarschaft := RQ(o, ε);
  if |Nachbarschaft| ≥ MinPts then // o ist ein Kernobjekt
    for i from 1 to |Nachbarschaft| do
       p := Nachbarschaft.get(i);
       if p.ClId in {UNCLASSIFIED, NOISE} then
              if p.ClId = UNCLASSIFIED then
                     füge p zur Menge seeds hinzu;
              p.ClId := ClusterId;
  entferne o aus der Menge seeds;
return true;
```

Diskussion dichtebasiertes Clustering

- Anzahl der Cluster wird automatisch bestimmt
- Parameter ε und MinPts sind relativ unkritisch (zumindest bei OPTICS)
- Komplexität ist $O(n^2)$ im allgemeinen Fall
- Dichtebasierte Verfahren benötigen nur ein Distanzmaß zur Anwendung
- Sowohl DBSCAN als auch OPTICS erlauben Noise-Objekte, die zu keiner Gruppe gehören
- DBSCAN und OPTICS sind reihenfolgeabhängig (Randpunkte)
- OPTICS ist zur visuellen Analyse geeignet
- Keine Zielfunktion wird optimiert
- Kein kompaktes Clustermodell
- Keine Zuordnungsanweisung für neue Objekte (unsupervised classification)

Outlier Detection

Definition nach Hawkins [Hawkins 1980]:

"Ein Outlier ist eine *Beobachtung*, die sich von den anderen *Beobachtungen* so deutlich unterscheidet, dass man denken könnte, sie sei von einem anderen Mechanismus generiert worden."

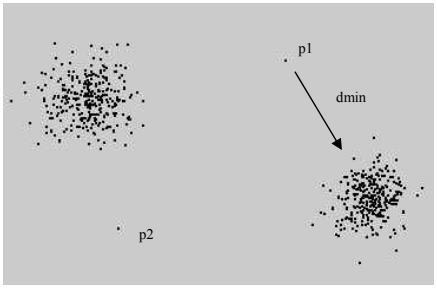
Was meint "Mechanismus"?

- Intuition aus der Statistik: "erzeugender Mechanismus" ist ein (statistischer) Prozess.
- Abnormale Daten (Outlier) zeigen eine verdächtig geringe Wahrscheinlichkeit, aus diesem Prozess zu stammen.
- Zusammenhang mit dem Clustering: Wenn das Clustering die Verteilung der Daten im Raum beschreibt, dann sind Outlier Objekte die nicht oder nur sehr schlecht Clustern zugeordnet werden können:
  - Maximaler Abstand zu allen Cluster Zentren (Part. Clustering)
  - Noise Punkte im dichtebasiertem Clustering

Beispiel: Distanzbasierte Outlier

- Definition *„(pct,dmin)*-Outlier" [Knorr, Ng 97]
  - Ein Objekt *p* in einem Datensatz *DB* ist ein *(pct,dmin)*-Outlier, falls mindestens *pct* - Prozent von Objekten aus *DB* eine größere Distanz als *dmin* zu *p* haben.

- Wahl von *pct* und *dmin* wird einem Experten überlassen.



- Beispiel: $p_1 \in DB$, *pct=0.95, dmin=8*

- p_1 *ist (0.95,8)-Outlier* => *95%* von Objekten aus *DB* haben eine Distanz > *8* zu p_1

Frequent Pattern Mining

Bisher: Muster beschreiben Menge aus vollständigen Objektbeschreibungen.

Idee: Muster können auch nur aus Teilen der gesamten Objektbeschreibung bestehen.

Hier: Muster ist ein bestimmter Teil der Objektbeschreibung, der in vielen Objekten der Datenmenge vorkommt.

Beispiel: Gruppenzusammensetzungen in MMORPG:

Grp1	Priester, Magier, Druide, Schurke, Krieger
Grp3	Magier, Magier, Jäger, Priester, Krieger
Grp3	Priester, Priester, Paladin, Druide, Krieger
Grp4	Hexer, Paladin, Schamane, Krieger, Priester
Grp5	Priester, Krieger, Jäger, Schurke, Krieger

→ Krieger, Priester,?,?,?

Varianten des Frequent Pattern Minings

- **Frequent Itemset Mining** (=> KDD1 und auf den nächsten Folien)
  Datenobjekte sind Teilmengen einer Menge von Items
  => Bestimme Teilmengen, die in vielen Objekten vorkommen

- **Frequent Substring Mining** (=> nächstes Kapitel)
  Datenobjekte sind Strings über einem endlichen Alphabet
  => Bestimme häufige Substrings

- **Frequent Subgraph Mining** (=> KDD2)
  Datenobjekte sind Graphen mit kategorischen Labeln/Knotentypen
  => Bestimme häufige isomorphe Teilgraphen

- **Regel Mining**: Auf Mustern lassen sich Regeln ableiten:
  Wenn Teilmuster A vorhanden ist,
  dann ist auch die Erweiterung A+B vorhanden

Grundlagen Frequent Itemsets (1)

- **Items** $I = \{i_1, ..., i_m\}$ eine Menge von Literalen
  z.B. Waren/Artikel bei einem Einkauf
- **Itemset** X: Menge von Items $X \subseteq I$
  z.B. ein kompletter Einkauf
- **Datenbank** *DB*: Menge von *Transaktionen T* mit $T = (tid, X_T)$
  z.B. Menge aller Einkäufe (= Transaktionen) in einem bestimmten Zeitraum
- **Transaktion** T enthält Itemset X: $X \subseteq T$
- Items in Transaktionen oder Itemsets sind **lexikographisch** sortiert:
  Itemset $X = (x_1, x_2, ..., x_k)$, wobei $x_1 \leq x_2 \leq ... \leq x_k$
- **Länge des Itemsets**: Anzahl der Elemente in einem Itemset
- ***k*-Itemset**: ein Itemset der Länge k
  {Butter, Brot, Milch, Zucker} ist ein 4-Itemset
  {Mehl, Wurst} ist ein 2-Itemset

Grundlagen Frequent Itemsets

- **Cover** eines Itemsets X: Menge der Transaktionen T, die X enthalten:
  $cover(X) = \{tid \mid (tid, X_T) \in DB, X \subseteq X_T\}$
- **Support des Itemsets** X *in* DB: Anteil der Transaktionen in DB, die X enthalten:

  $support(X) = |cover(X)|$

  Bemerkung: $support(\varnothing) = |DB|$
- **Häufigkeit eines Itemsets** X *in* DB:

  Wahrscheinlichkeit, dass X in einer Transaktion $T \in DB$ auftritt:
  $frequency(X) = P(X) = support(X) / |DB|$
- **Häufig auftretendes (frequent) Itemset** X *in* DB:

  $support(X) \geq s \qquad (0 \leq s \leq |DB|)$

  s ist ein absoluter support-Grenzwert

Alternativ: $frequency(X) \geq s_{rel}$ wobei $s = \lceil s_{rel} \cdot |DB| \rceil$

Problemstellung: Frequent Itemset Mining

Gegeben:

- Eine Menge von Items I
- Eine Transaktionsdatenbank DB über I
- Ein absoluter support-Grenzwert s

Aufgabe: Finde alle frequent Itemsets in DB, d.h. $\{X \subseteq I \mid support(X) \geq s\}$

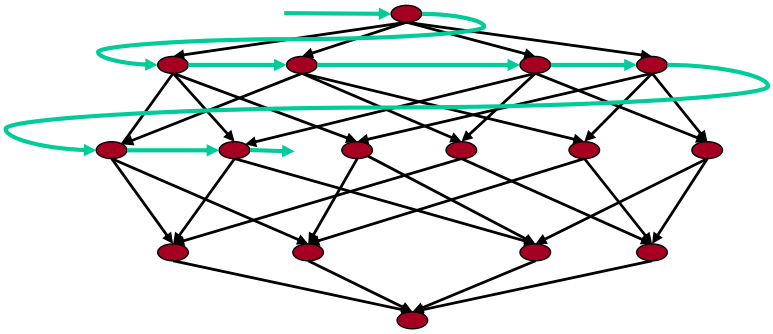
TransaktionsID	Items
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F

Support der 1-Itemsets:
(A): 75%, (B), (C): 50%, (D), (E), (F): 25%,
Support der 2-Itemsets:
(A, C): 50%,
(A, B), (A, D), (B, C), (B, E), (B, F), (E, F): 25%

Itemset Mining

A priori Algorithmus [Agrawal & Srikant 1994]

- zuerst die einelementigen Frequent Itemsets bestimmen, dann die zweielementigen und so weiter (Breitensuche)



- Finden von k+1-elementigen Frequent Itemsets:
  - Nur solche k+1-elementigen Itemsets betrachten, für die alle k-elementigen Teilmengen häufig auftreten
- Bestimmung des Supports durch Zählen auf der Datenbank (ein Scan)

A priori Algorithmus

C_k: die zu zählenden Kandidaten-Itemsets der Länge k

L_k: Menge aller häufig vorkommenden Itemsets der Länge k

```
Apriori(I, DB, minsup)
L1 := {frequent 1-Itemsets aus I};
k := 2;
while Lk-1 ≠ ∅ do
    Ck := AprioriKandidatenGenerierung(Lk-1);
    for each Transaktion T ∈ DB do
        CT := Subset(Ck, T);// alle Kandidaten aus Ck, die
                            // der Transaktion T enthalten sind;
        for each Kandidat c ∈ CT do c.count++;
    Lk := {c ∈ Ck | c.count ≥ minsup};
    k++;
return ∪k Lk;
```

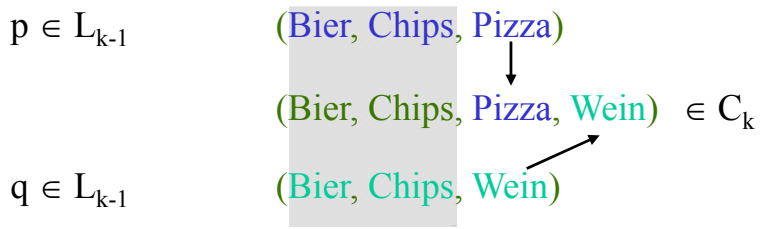
Kandidatengenerierung im Apriori Algorithmus (1)

Anforderungen an Kandidaten-Itemsets C_k, Obermenge von L_k

- wesentlich kleiner als die Menge aller k-elementigen Teilmengen von I

Schritt 1: Join

- k-1-elementige Frequent Itemsets p und q
- p und q werden miteinander verbunden, wenn sie in den ersten k-2 Items übereinstimmen



Kandidatengenerierung im Apriori Algorithmus (2)

Schritt 2: Pruning

Entferne alle Kandidaten-k-Itemsets, die eine k-1-elementige Teilmenge enthalten, die nicht zu L_{k-1} gehört

Beispiel:

$L_3 = \{(1\ 2\ 3), (1\ 2\ 4), (1\ 3\ 4), (1\ 3\ 5), (2\ 3\ 4)\}$

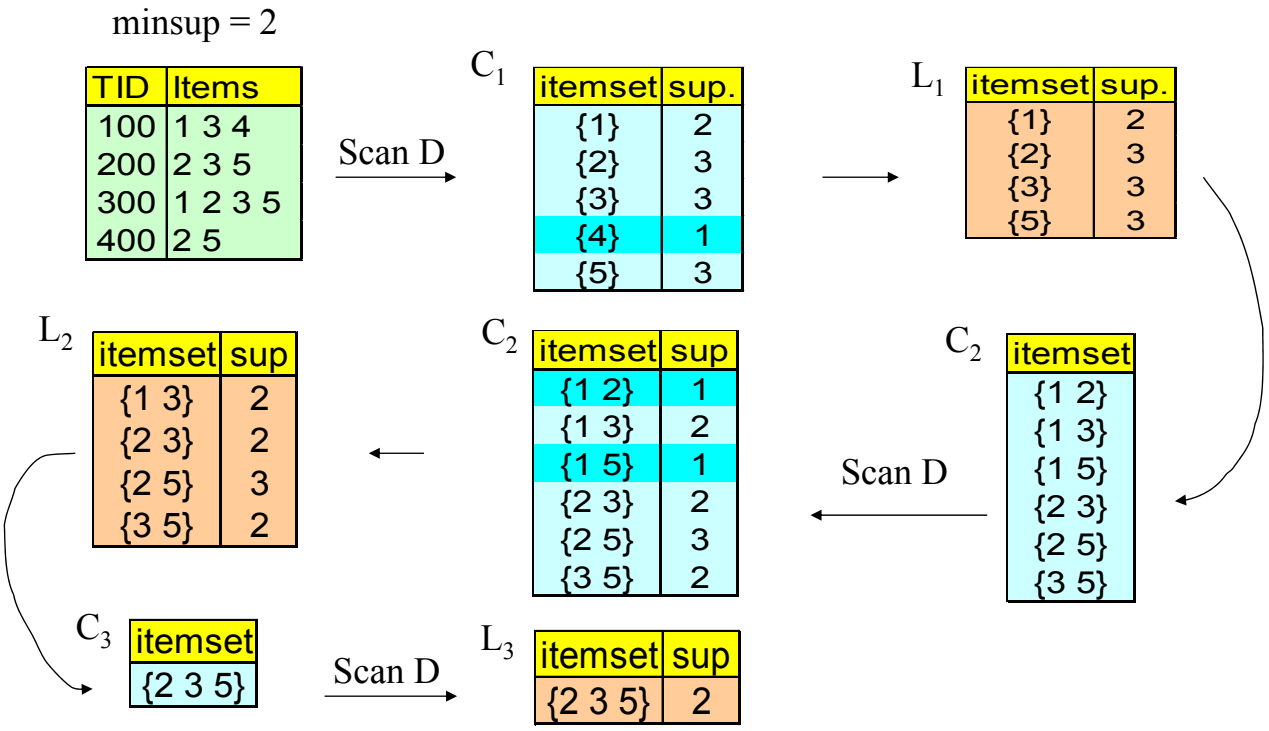
nach dem Join-Schritt: Kandidaten = $\{(1\ 2\ 3\ 4), (1\ 3\ 4\ 5)\}$

im Pruning-Schritt:

lösche (1 3 4 5)

⇨ $C_4 = \{(1\ 2\ 3\ 4)\}$

Beispiel für den Apriori Algorithmus



Frequent Itemset Mining

Eigenschaften:

- Benötigt für alle Itemsets der Länge *k* einen Datenbank-Scan

  $\Rightarrow O(l \cdot |D|)$

- Menge der generierten Kandidaten, die nicht frequent sind entspricht dem negativen Rand

  $\Rightarrow O\left(\binom{m}{\lfloor m/2 \rfloor}\right)$ (Sperners Theorem)

- Wenn nicht alle Kandidaten Itemsets in den Hauptspeicher passen, werden Kandidaten blockweise auf min. Support überprüft

Lernziele

- Was ist Data Mining und KDD?
- Welche Schritte hat der KDD-Prozess?
- Was ist Supervised Learning?
  - Linear Regression
  - kNN Klassifikation
  - Bayes Learner
  - Bewertung von Supervised Verfahren
- Was ist Unsupervised Learning
  - Clustering (partitionierende Verfahren, DBSCAN, OPTICS)
  - Outlier Detection (Distanzbasierte Outlier)
  - Frequent Pattern Mining (Frequent Itemset Mining mi Apriori-Alg.)

Literatur

- **Skripte zu KDD I**:
  http://www.dbs.ifi.lmu.de/cms/Knowledge_Discovery_in_Databases_I_(KDD_I)
- Ester M., Sander J.:
  Knowledge Discovery in Databases: Techniken und Anwendungen
  Springer Verlag, September 2000.
- Han J., Kamber M., Pei J.:
  Data Mining: Concepts and Techniques
  3rd edition, Morgan Kaufmann Publishers, March 2011.
- H.-P. Kriegel, M. Schubert, A. Zimek
  Angle-Based Outlier Detection in High-dimensional Data
  In Proceedings of the 14th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Las Vegas, NV: 444–452, 2008.
- M. Ankerst, M. M. Breunig, H.-P. Kriegel, J. Sander
  OPTICS: Ordering Points To Identify the Clustering Structure
  In Proceedings of the ACM International Conference on Management of Data (SIGMOD), Philadelphia, PA: 49–60, 1999.