Skript zur Vorlesung
Managing and Mining Multiplayer Online Games
im Sommersemester 2012

Kapitel 7: Knowledge Discovery und Data Mining in a Nutshell

http://www.dbs.informatik.uni-muenchen.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]

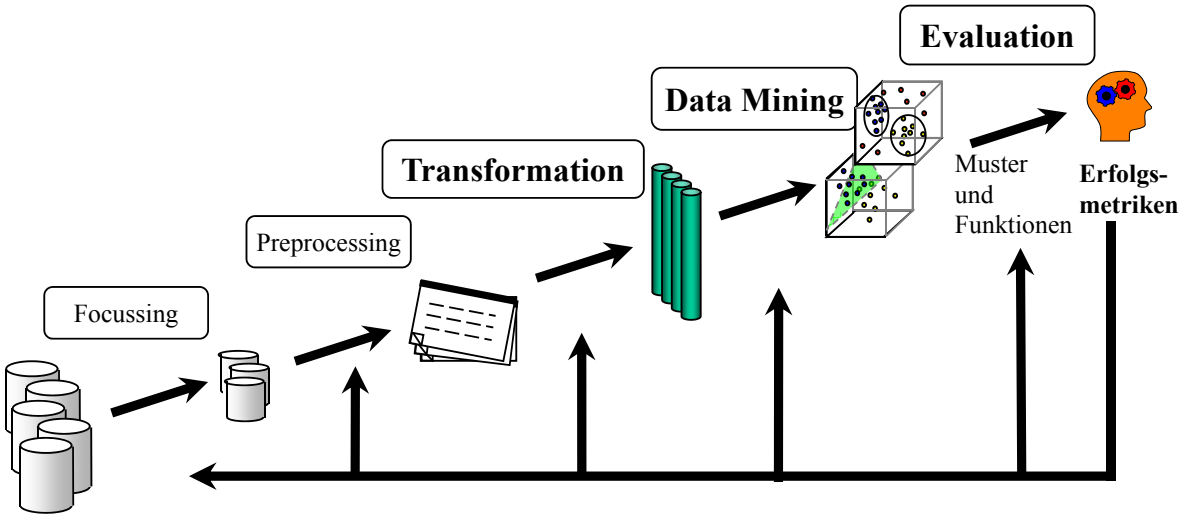
Knowledge Discovery in Databases (*KDD*) ist der Prozess der (semi-) automatischen Extraktion von Wissen aus Datenbanken, das

- *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 weiteroptimiert (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 Mittschnitt des TCP-Verkehrs, um eine Vorhersagemodell zu trainieren, das erkennt ob Spieler von einem Bot gesteuert wird.

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

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

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.
  - Vergleich 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 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 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 statitisch 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 das 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)
- **Modelkomplexität**: 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 des 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 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 Objektedarstellung

Feature-Vektoren: Standarddarstellung für die meisten Algorithmen

Grundidee:

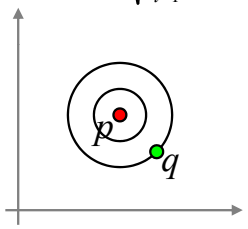
- **Feature**: Merkmal, das das Objekt beschreibt.
  Beispiel: durchschnittliche Anzahl von Paketen pro Sekunde
- **Wertebereiche**:
  - Nominal: Gleichheit und Ungleichheit feststellbar (Beispiel: Name)
  - Ordinal: Werte unterliegen einer Ordnung (Beispiel: Gildenrang)
  - Numerisch: Werte haben einen 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):

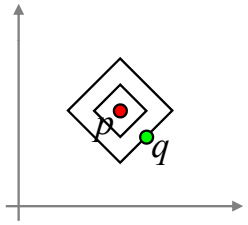
$$d_2(p,q) = \sqrt{\sum_{i=1}^{d} (p_i - q_i)^2}$$



Natürlichstes Distanzmaß

Manhattan-Norm (L_1):

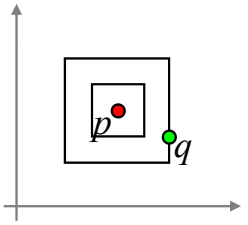
$$d_1(p,q) = \sum_{i=1}^{d} |p_i - q_i|$$



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

Normen, Ähnlichkeiten und Kernel

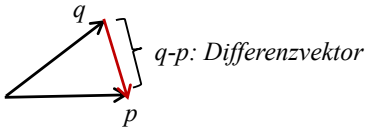
- *Euklidische Distanz*: Länge des Differenzvektors
- *Länge eines Vektors*: Norm des Vektors
- *Norm*: Skalarprodukt mit sich selbst
- Eigenschaften eines Skalarprodukts

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

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

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

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



$\|q - p\|$

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

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

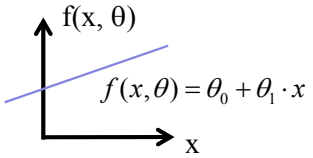
Absoluter Fehler: $L_{abs}(f,T) = \sum_{o \in T} \left(o.c = f(o.v)\,?1:0\right)$

Training von Supervised Methoden

- Art der Funktion f ist idR. 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:

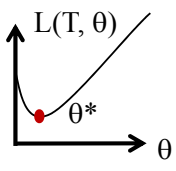
$$f(x,\theta) = \theta_0 + \sum_{i=1}^{d} \theta_i \cdot x_i$$



Lösung im 2D-Raum

Training: Minimiere die Fehlerfunktion

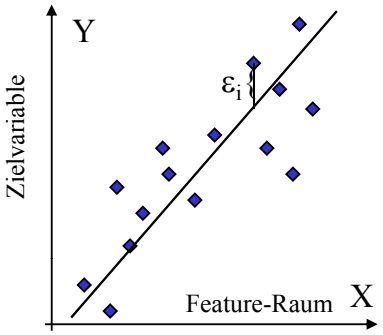
- Fehler Funktion L berechnet den Fehler den f für T macht
- Wähle die Parameter θ*, die L minimieren
- ***Vorgehen***: Differenzieren der Fehlerfunktion nach θ und Suche nach dem Minimum θ*.
  => Fehlerfunktion sollte konvex sein
  (nur ein Extremum und das ist ein Minimum)
  => allgemeine Fehlerfunktionen 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_o: $\frac{\partial L^2}{\partial \theta_0} = \sum_{(x,y)\in T} (2\theta_0 - 2y + 2\theta_1 x) = 2\left(\theta_0 |T| - \sum_{(x,y)\in T} y + \theta_1 \sum_{(x,y)\in T} x\right)$

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

Ableitung nach θ_0:

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

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

weitere Anmerkungen zum Supervised Learning

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

  Beispiel: lineare Ridge-Regression

  Regularisierungsterm

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

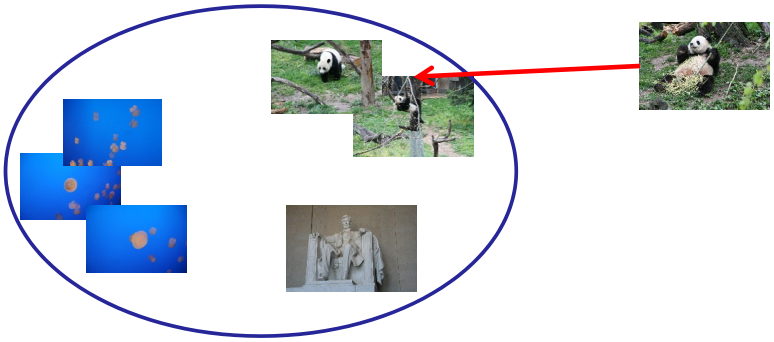
- Weitere Problemformulierungen basieren häufig auf Optimierungsproblemen mit Nebenbedingungen und sind daher wesentliche aufwendiger (quadratischer Programme, semidefinite Programme,..)
- Zielfunktionen müssen nicht unbedingt konvex sein (Neuronale Netze)
  => Optimierung findet evtl. nur lokale Extrema
- 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 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 mal 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 W‘keit 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)*
(W'keit: das *B* spielt unter der Beobachtung *v*)

- Aus der Annahme folgt das $p(v) = p(A)p(v|A) + p(B)p(v|B)$
  wobei *P(B)* bzw. *P(A)* beschreibt wie wahrscheinlich *B* bzw. *A* wirkliche spielen. (Apriori 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: $P(B \mid v) = \frac{P(B)P(v \mid B)}{P(v)}$
- 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

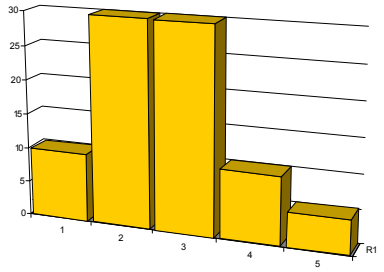
- Apriori Wahrscheinlichkeiten *P(c)* werden idR durch relative Häufigkeiten der Klassen in der Trainingsmengen*T* beschrieben.
  (17 von 100 Mitschnitten sind Bots: *P(B)* =17%)
- Der Bestimmung von *p(v|c)* liegen Verteilungsmodelle zugrunde
  Wobei sich die Modelle für die $c \in C$ meist nur durch Parameter unterscheiden.
- Training über Berechnnung der relativen Häufigkeiten aus den Trainingsdaten die zu Modell c gehören.

Beispiel: Betrachte 2 Würfel

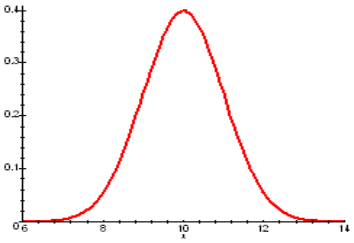
- Mögliche Ereignisse: {1,2,3,4,5,6}
- Würfel W1 ist gleichverteilt: 1/6 für 1 bis 6
- Würfel W2 Verteilung: 1:1/12, 2:1/12: 3:1/6, 4:1/6, 5:1/6, 6:1/3
- *p(v=1|W1) = 1/6, p(v=6|W2)=1/3*

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 $p(v_1,..,v_d|\ c)$ berechnet werden.

Problem: Sind $v_1,..,v_d$ abhängig voneinander ?

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

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

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

  Vorteil: Berechnung stimmt selbt bei unabhängigen Parametern.

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

Statistische Modelle

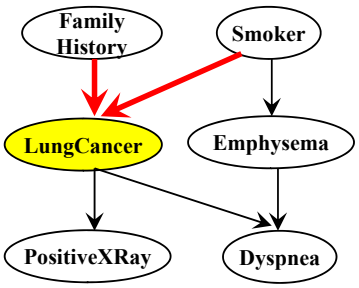
Fortgeschrittene Lösungen:

=> 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 Zielvariabler sind in der Regel Mangelwahre. (manuelle 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 1 werden n/k Datenobjekte (fold) aus der Trainingsmenge in die Testmenge verschoben und getestet.

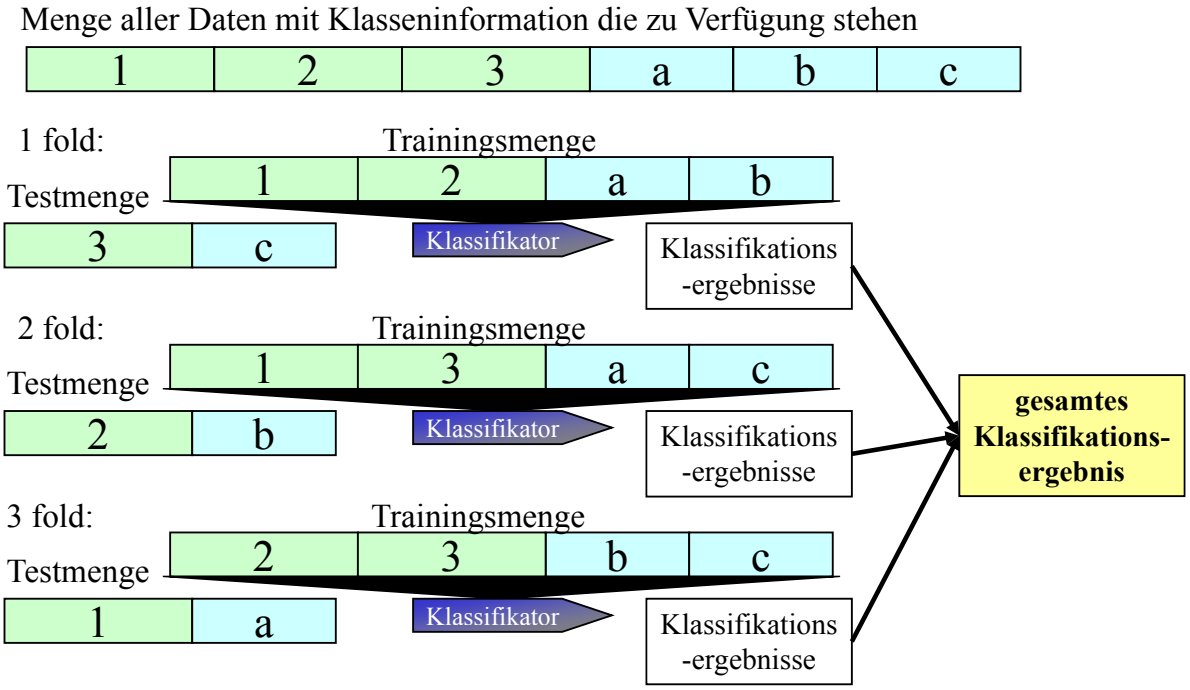
- **Stratifizierung**: In jeder Teilmenge (fold) sollte möglichst die gleiche Verteilung über die Klassen gelten, wie in der gesamten Datenmenge. (besserer Erhalt der Apriori Wahrscheinlichkeit)

- 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 ...	Klasse1	Klasse 2	Klasse 3	Klasse 4	other
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
other	3	1	9	15	13

korrekt klassifizierte Objekte

Aus der Konfusionsmatrix lassen sich u.a. folgende Kennzahlen berechnen :
Accuracy, Classification Error, 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)

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

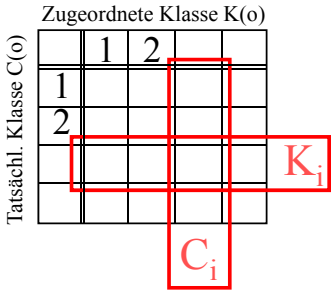
- *Beobachteter Klassifikationsfehler* (apparent classification error)

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

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

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