

Analysetechniken Zusammenfassung

Freitag, 26. Januar 2018 01:12

Überblick

- Wichtige Data-Mining Probleme
 - Association Rules
 - zB Market Basket Analysis: Wie oft legen Kunden Waren relativ häufig in Einkaufswagen?
 - Clustering
 - Nah beieinander liegende Items gruppieren
 - Varianten: Unterschiedlichen Attributtypen (Zahlen, Enums, Mengenwertig...), Vielfalt der Cluster im Datenbestand (Unterschiedliche Form, Dichte, Größe, ...), Clusterzugehörigkeit nicht apriori bekannt (supervised vs unsupervised)
 - Supervised: Clustergrenzen vorher nicht bekannt, Unsupervised: Klassenzugehörigkeiten des Trainingsdatenbestands bekannt (=> Klassifikation)
 - Klassifikation
 - Item mit n Attributen, man will $n+1$ Attribut vorhersagen. Grundlage: Trainingsmenge mit $n+1$ korrekten Werten
 - zB durch Entscheidungsbaum
 - Linear Classifier
 - n Attribute X_i , Schwellenwert T
 - Es werden n Gewichte $w \in \mathbb{R}^n$ berechnet
 - x ist positive Instanz gdw $xw > T$
 -

Attribute 1, ..., n		Attribut n+1
Alter	Gehalt	Kreditwürdigkeit
...

.kreditwürdig' gdw. $0.3 \cdot \text{Alter} + 2 \cdot \text{Gehalt} + \dots > T$

Erfasster Bildschirmausschnitt: 02.02.2018 18:46

□ Belanglose Attribute erhalten so kleines Gewicht, für Klassifikation hilfreiche Attribute dafür großes Gewicht

□ Sinnvoll für reelle und boolsche Werte

□ 1-Rules:

◆ Jedes Attribut für sich betrachten, Vorhersagen testen und dann das Attribut wählen, mit dem die Fehlerquote am kleinsten ist

Evaluation der Attribute der Wetter-Daten				
	Attribute	Rules	Errors	Total Errors
1	outlook	sunny → no	2/5	4/14
		overcast → yes	0/4	
		rainy → yes	2/5	
2	temperature	hot → no*	2/4	5/14
		mild → yes	2/6	
		cool → yes	1/4	
3	humidity	high → no	3/7	4/14
		normal → yes	1/7	
4	windy	false → yes	2/8	5/14
		true → no*	3/6	

*Bei gleich wahrscheinlichen Ausgängen zufällige Auswahl.

Erfasster Bildschirmausschnitt: 02.02.2018 18:49

□ Overfitting

◆ zB durch Kreditkartennummern

□ Beispieldienstleistungen

◆ Fehlerlokalisierung: Softwarefehler auffinden. Occasional Non-terminating Bugs

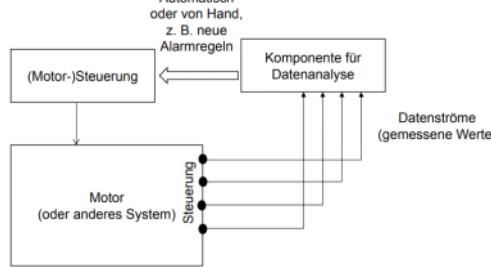
◊ Verschiedene Ausführungen des Programms generieren (als Graphs) und korrekt/fehlerhaft markieren

◊ Graph Mining Methoden um Muster zu finden

◆ Predictive Maintenance

◊ Für zB Motoren Zeitpunkt von Schadensfall vorhersagen

◊ Zwei Fehlerfälle: Ausfall vorhergesagt aber nicht passiert, Ausfall nicht vorhergesagt aber passiert



Erfasster Bildschirmausschnitt: 02.02.2018 18:53

◊ Modellorientiert (Experte formuliert Fehlerzustände manuell) oder Datenorientiert (Musterbehaftete Fehler, diese Vorlesung)

▶ Voraussetzung für Datenorientierten Aspekt

- Verhalten folgt Muster
- Großer Trainingsdatenbestand
- Messdaten und vorherzusagende Daten müssen zusammengebracht werden
- Infrastruktur muss relevante Phänomene erfassen

▶ Statische Betrachtung: Jeder Zeitpunkt für sich

▶ Dynamische Betrachtung: Zeitliche Entwicklung wird berücksichtigt

▶ Change Detection (univariant oder multivariant = mehrere Dimensionen)

Statistische Grundlagen

- Datenbeschaffung
 - Nominal, Ordinal, Diskret, Kontinuierlich
 - Dimensionalität: Eindimensional (univariant) oder Multidimensional (multivariant), Hochdimensional (sehr viele Dimensionen), keine Dimensionalität

- Metrische Daten:

- In metrischen Räumen gegeben: Domäne M und Metrik d mit folgenden Eigenschaften für alle Objekte p, q, r ∈ M:
- Symmetrie: $d(p, q) = d(q, p)$
 - Definitheit: $d(p, q) = 0 \Leftrightarrow p = q$
 - Dreiecksungleichung: $d(p, r) \leq d(p, q) + d(q, r)$

Erfasster Bildschirmausschnitt: 02.02.2018 19:00

- Datenstreuung charakterisieren

- zB durch median, max, min, Quantile, Outlier, Varianz, usw
- Aggregate (count, sum, min, max, avg)

- Distributive Aggregatsfunktion: Es gibt eine Funktion G mit

$$F(\{X_{i,j}\}) = G(\{F(\{X_{i,j} | i=1, \dots, I\}) | j=1, \dots, J\}),$$

Also: Funktion kann auf Teilmengen durchgeführt werden, Teilmengenergebnisse werden von G ausgewertet

Beispiele: min, max, count

- Algebraische Aggregatsfunktion

Es gibt Funktion G, die M-Tupel liefert, und H,
so dass $F(\{X_{i,j}\}) = H(G(\{X_{i,j} | i=1, \dots, I\}) | j=1, \dots, J)$,
M ist a priori bekannt, ebenso der Typ der Tupel.

Beispiel: avg()

Also: G kann auf Teilmengen durchgeführt werden, Teilmengenergebnisse werden von H ausgewertet

- Holistische Aggregatsfunktion

Man kann keine Beschränkung des Speicherbedarfs
für Sub-Aggregate, d. h. für die Aggregate

über die $\{X_{i,j} | i=1, \dots, I\}$, angeben.

Beispiel: häufigsterWert(), median()

- Distributive und Algebraische Funktionen können parallel oder schrittweise durchgeführt werden

- Self-Maintainable Aggregatsfunktion: Nach Änderung der Daten kann der neue Wert der AF aus altem Wert und Änderungsdelta berechnet werden

- Qualifizierung der Daten

- Mittel:

$$\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^n x_i \quad \bar{x}_w = \frac{\sum_{i=1}^n w_i \cdot x_i}{\sum_{i=1}^n w_i}$$

Erfasster Bildschirmausschnitt: 03.02.2018 16:40

"Durchschnitt", algebraisch.

- Mitte: $(\max - \min)/2$

- Median: Mittlerer Wert bei sortierter Anordnung aller Werte der in der Mitte (bzw Durchschnitt der beiden mittleren Werte). Holistisch.
□ Nicht nur anwendbar auf numerische, sondern auch ordinale Daten

- Modalwert/Modus: Wert, der im Datenbestand am häufigsten vorkommt. Unimodal, bimodal, Trimodal. Undefiniert für einzigartige Werte

- Quantifizierung der Streuung der Daten

- Quartile: Q_1 (Oberste 25%), Q_3 (Oberste 75%)

 - IQR: Inter Quartile Range = $Q_3 - Q_1$

 - Fünf gleichmäßig verteilte Werte: min, Q_1 , median, Q_3 , max

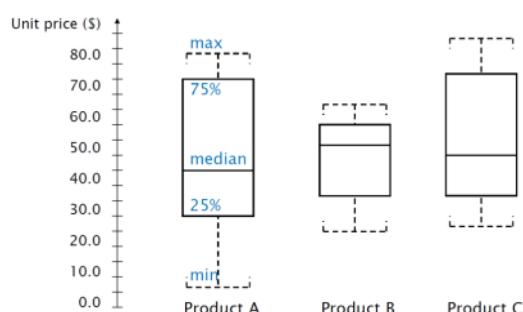
 - Ausreißer: Werte, die mehr als $1.5 * \text{IQR}$ von Q_1 nach unten bzw von Q_3 nach oben abweichen.

- Varianz:

$$\sigma^2 = \frac{1}{n-1} \cdot \sum_{i=1}^n (x_i - \bar{x})^2$$

Algebraisch. Standardabweichung $\sigma = \sqrt{\sigma^2}$

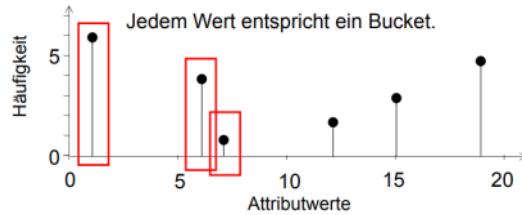
- Boxplots



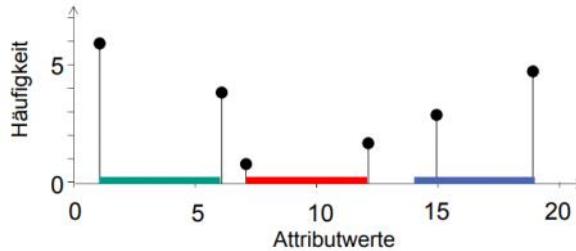
Erfasster Bildschirmausschnitt: 03.02.2018 16:46

- Histogramme

 - Normale Histogramme

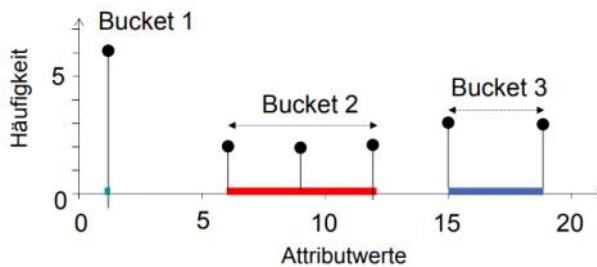


Erfasster Bildschirmausschnitt: 03.02.2018 16:47
 Equi-Width Histogramme



Attributwerte	[1, 6]	[7, 12]	[14, 19]
Häufigkeit	$6 + 4 = 10$	$1 + 2 = 3$	$3 + 5 = 8$

Erfasster Bildschirmausschnitt: 03.02.2018 16:47
 Equi-Depth Histogramme



Attributwerte	1	[6, 12]	[15, 19]
Häufigkeit	6	$3 \times 2 = 6$	$2 \times 3 = 6$

Erfasster Bildschirmausschnitt: 03.02.2018 16:48

Probleme: Nur wenige Dimensionen, keine Antwortverfeinerung, schlechte Approximationsgenauigkeit

▪ Entropie

Entropie einer Menge: Wie zufällig sind die Daten der Menge verteilt? Maß für Unordnung.

$$H(S) = -\sum_j p_j \cdot \log p_j$$

p_j = relative Häufigkeit der Klasse j in S

Logarithmanteil ist statistische Signifikanz: Wie überraschend ist Ereignis?

Entropie ist minimal für $p_1=1$, maximal für $p_i=p_j$

▪ Wahrscheinlichkeitstheorie

[TODO]

Unterschied Kovarianz zu Korrelationsmaß: Kovarianz ist nicht normiert und damit in verschiedenen Kontexten nicht vergleichbar, Korrelationsmaß ist zwischen -1 und 1.

▪ Statistische Tests

Chi-Quadrat Test

◆ Sind zwei Verteilungen unabhängig voneinander?

◆ Ordinale oder diskrete Ereignisse zB durch Histogramme beschrieben.

◆ Vorgehen:

◊ Beobachtete Anzahlen von Ereignissen zusammenragen und mit erwarteten vergleichen

◊ Prüfgrößen berechnen, die Abweichungen für alle Einzelereignisse aggregiert

◊ Wahrscheinlichkeit dieser Prüfgröße bei Unabhängigkeit ermitteln aus Chi-Quadrat Verteilung

◊ Hypothese, dass Verteilungen unabhängig sind, wird zurückgewiesen, wenn $p \leq \alpha$ (Schwellwert, zB = 0.01)

Kolmogorov-Smirnov Test

◆ Stimmen zwei Wahrscheinlichkeitsverteilungen überein?

◆ Nicht zwangsweise ordinale oder diskrete Ereignisse

◆ Konzeption:

◊ Nullhypothese $H_0: F(x) = F_0(x)$ soll verworfen werden, Vergleich F_0 mit empirischer Verteilungsfunktion F_n

Kolmogorov-Smirnov-Test (2)

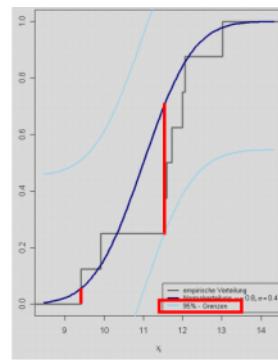
i x_i $S(x_i)$ $F_0(x_i)$ $S(x_{i-1}) - F_0(x_i)$ $S(x_i) - F_0(x_i)$



Kolmogorov-Smirnov-Test (2)

i	x_i	$S(x_i)$	$F_o(x_i)$	$S(x_{i-1}) - F_o(x_i)$	$S(x_i) - F_o(x_i)$
1	9,41	0,125	0,056	-0,056	0,069
2	9,92	0,250	0,140	-0,015	0,110
3	11,55	0,375	0,709	-0,459	-0,334
4	11,60	0,500	0,726	-0,351	-0,226
5	11,73	0,625	0,767	-0,267	-0,142
6	12,00	0,750	0,841	-0,216	-0,091
7	12,06	0,875	0,855	-0,105	0,020
8	13,02	1,000	0,978	-0,103	0,022

■ $H_0: F(x) = F_o(x) = \Phi(x|11; 1)$



Beispiel ist aus Wikipedia.

■ Erläuterung:

- Empirische Werte sind sortiert. Verteilungsfunktion in grau.
- Erwartung: y-Wert des ersten Messpunkts liegt zwischen 0 und 0,125.
- Abweichung dieses y-Werts von 0 bzw. 0,125 ist in Tabelle eingetragen.
- In Tabelle nachschauen, welche Abweichungen noch akzeptabel sind.

Erfasster Bildschirmausschnitt: 03.02.2018 17:04

- ◆ Maximale Distanz zwischen kumulativen Häufigkeitsverteilungen ist Entscheidungsgrundlage
- ◆ Beispiel, konstruiert: Angenommen, alle Frauen verdienen in etwa gleich viel; Spread bei Männern ist hingegen groß. Dann sind Verteilungen (für Männer und Frauen) offensichtlich unterschiedlich. (Gesellschaftlich) interessante Frage aber: Verdienen Männer mehr als Frauen?

□ Wilcoxon Mann Whitney Test

- ◆ Zwei Verteilungen mit Verteilungsfunktionen F_x und F_y mit $F_y(x) = F_x(x-a)$
- ◆ Unabhängige Stichproben X_1, \dots, X_m und Y_1, \dots, Y_n
- ◆ $H_0: a = 0; H_1: a \neq 0$
- ◆ Ist Abweichung der Mediane statistisch signifikant? (dagegen Kolmogorov; Stimmen zwei Verteilungen überein?)
- ◆ Sortiere Items nach Wert, sodass Items Ränge haben.
 - ◊ Rangsummenstatistik: Ränge der Kategorien aufsummieren, daraus Kennzahl berechnen und mit Tabelleneintrag vergleichen

□ Bernoulli Experiment (kein statistischer Test)

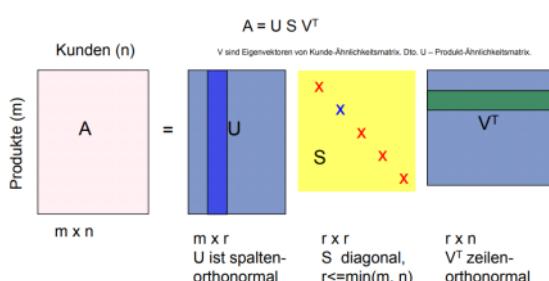
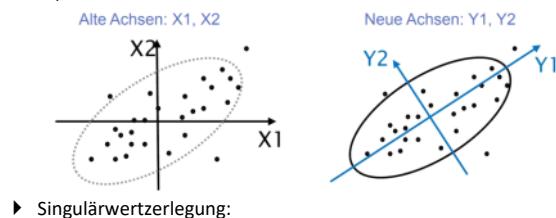
- ◆ N Datenobjekte, Erfolgswahrscheinlichkeit p eines einzelnen Experiments (zB korrekte Klassifizierung), S erfolgreiche Experimente (succeses)
- ◆ Erfolgsquote $f=S/N$ ist Zufallsvariable, Varianz= $p(1-p)/N$
- ◆ Gegeben beobachtete Erfolgsquote, wie groß ist p wahrscheinlich? (?)
 - ◊ $\Pr[-z \leq X \leq z] = c$. Gegeben c, wie groß ist z?
 - ◊ Meist nur oberer Teil wegen Symmetrie der Normalverteilung: $\Pr[X \geq z]$

□ Forschung: Wie statistische Tests effizient in Datenbanken implementieren?

- Vorteile von statistischen Tests
 - ◆ Nur Werkzeug für Anwendungsentwickler
 - ◆ Performanz
 - ◆ Gute Kombinationsmöglichkeiten mit anderen DB Features (SQL Queries)

■ Datenreduktion

- Dimensionality Reduction: Reduziere Anzahl der Attribute
- Diskretisierung: Weniger mögliche Werte pro Attribut, Vergrößerung
- Numerosity Reduction: Reduziere Anzahl der Datenobjekte
 - ◆ Parametrische Verfahren: Datenverteilung folgt bestimmten Modell, schätzt Modellparameter, speichere nur diese evtl mit Ausreißern
 - ◆ Nichtparametrische Verfahren: Sampling, Clustering, Histogramme
 - ◊ Sampling: Arbeit auf *repräsentativem* Ausschnitt des Datenbestands
 - ◊ Feature Selection: Auswahl einer Teilmenge von Attributen
 - ▶ Per Hand durch Domänenexperte oder automatisches Auswahlkriterium: Werte fehlender Attribute lassen sich mittels vorhandener Werte vorhersagen
 - ▶ Vorhersagekraft jedes Attributes ermitteln, beste Attribute behalten. Schrittweise Eliminierung
 - ◊ Hauptachsentransformation (Singulärwertzerlegung)
 - ▶ Gegeben N k-dimensionale Datenobjekte, finde c <= k orthogonale Vektoren (Hauptachsen), die Datenbesand gut repräsentieren



- ◊ Dimensionsreduktion: Weglassen der kleinsten Diagonalelemente (von S) und Sortieren der Reihenfolge der

- Koordinatenachsen absteigend
- ◊ Clustering: Finden von Ausschnitten des Raums mit überdurchschnittlich vielen Datenobjekten, dann nur Cluster speichern
 - ◊ Diskretisierung: Intervallgrenzen finden und Attributwerte vergrößern
 - Entropiebasierte Diskretisierung
 - Datenbestand S in S_1, S_2 partitionieren mit Abgrenzung T , sodass
$$Entropie_{Split}(S, T) = \frac{|S_1|}{|S|} \cdot Entropie(S_1) + \frac{|S_2|}{|S|} \cdot Entropie(S_2)$$

Erfasster Bildschirmausschnitt: 03.02.2018 17:27

- Diskretisierung top-down (wie eben) oder bottom-up, merge-basiert oder split-basiert.
 - Merge: Finde beste benachbarte Intervalle und führe sie zu größerem Intervall zusammen, bis Abbruchkriterium erfüllt
- ChiMerge: Jeder Wert ist ein Intervall, χ^2 Tests für alle Paare benachbarter Intervalle und ggf mergen.

Informatik Grundlagen: Räumliche Indexstrukturen

- Index
 - Seitenweise Anordnung von Daten im Hauptspeicher.
 - Potentiell für mehrere Attribute. Sortiert

■ Index für (gpa, name):

3.6			
((2.3, Chad), (2, 3))	((3, Mary), (2, 1))		
((2.5, Chang), (1, 2))	((3.1, James), (2, 2))		
((2.8, Lam), (3, 1))	((3.2, Tom), (1, 1))		
((2.8, Pat), (1, 4))	((3.5, Leila), (2, 4))		

■ Index für (name, gpa):

Lc			
((Bob, 3.7), (1, 3))	((James, 3.1), (2, 2))		
((Chad, 2.3), (2, 3))	((Kane, 3.8), (3, 3))		
((Chang, 2.5), (1, 2))	((Kathy, 3.8), (3, 2))		
((Chris, 3.9), (4, 1))	((Lam, 2.8), (3, 1))		

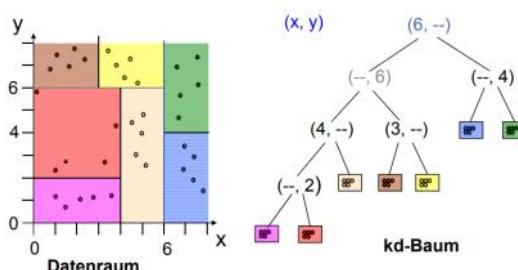
Tom, 20, 3.2, EE	Mary, 24, 3, ECE	Lam, 22, 2.8, ME	Chris, 22, 3.9, CS
Chang, 18, 2.5, CS	James, 24, 3.1, ME	Kathy, 18, 3.8, LS	Vera, 17, 3.9, EE
Bob, 21, 3.7, CS	Chad, 28, 2.3, LS	Kane, 19, 3.8, ME	Louis, 32, 4, LS
Pat, 19, 2.8, EE	Leila, 20, 3.5, LS	Martha, 29, 3.8, CS	Shideh, 16, 4, CS

- Räumliche Indexstruktur
 - Ähnlichkeit von Items => Suche nach nächstem Nachbarn
 - Attributnormalisierung: Unterschiedene Einheiten pro Dimension, um Abstand zu berechnen: Normalisierung

$$\text{Normalisierung: } a_i = \frac{v_i - \min v_j}{\max v_j - \min v_j}$$

- Beispiel: Wertegrenzen [0,100Jahre] und [0,500€] aufeinander anpassen

◦ Kd-Baum



Erfasster Bildschirmausschnitt: 04.02.2018 22:57

- Split-Richtung: Eine Dimension nach der anderen, dann wieder von vorne
- Im Beispiel 4 Splitrichtungen
- Anfragen für räumliche Indexstrukturen
 - Punktanfragen
 - Bereichsanfragen
 - NN-Anfragen

nächsterNachbar(kdB-Baum, Anfrage)

```

1 Queue ← neue Prioritätswarteschlange()
2 Region ← Wurzelknoten des kdB-Baums
3 Distance ← Abstand(Region, Anfrage)
4 Einfügen(Queue, Distance, Region)

5 WHILE (true) DO
6   Element ← Head der Queue
   // Dieser Schritt beinhaltet auch „pop“,
   // d. h. Entnahme des Queue-Heads
7   IF (Element ist Datenpunkt) THEN
8     BEGIN; Gib Element zurück; RETURN; END
9   ELSE
10    Traversierung(Element, Queue, Anfrage)
11 END WHILE

```

Traversierung(Element, Queue, Anfrage)

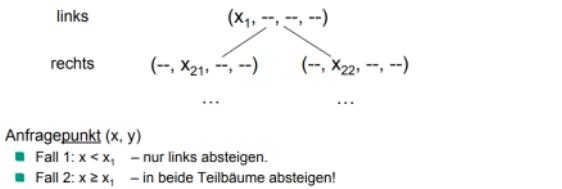
```

1 IF (Element ist Blatt) THEN
2 FOR EACH (Datenpunkt in Element) DO
3   Distance ← Abstand(Datenpunkt, Anfrage)
4   Einfügen(Queue, Distance, Datenpunkt)
5 END FOR
6 ELSE
7 FOR EACH (Kind von Element) DO
8   Distance ← Abstand(Kind, Anfrage)
9   Einfügen(Queue, Distance, Kind)
10 END FOR
11 END IF

```

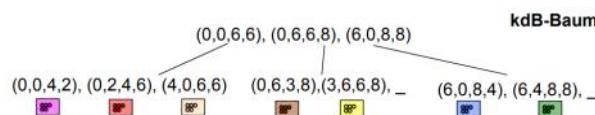
- Für alle: Siehe Folien für Beispielrechnung

- Objekte mit räumlicher Ausdehnung
 - Objekte sind Rechtecke, nicht Punkte
 - Minimum Bounding Rectangle



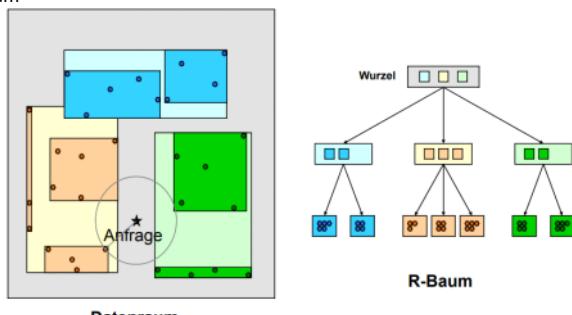
- Bereichsanfrage; Welche Rechtecke überlappen Anfragerechteck?
- Welche Rechtecke sind im Anfragerechteck enthalten?
- Welche Rechtecke enthalten Anfragerechteck?

- kdB-Baum: Balancierter kd-Baum



- Kombination von kd-Baum und B*-Baum
- Physischer Knoten enthält mehrere logische Regionen
- Physische Knoten nicht mehr pro Split Dimension
- Vorteile
 - ◆ Abstand Daten zu Wurzel immer gleich
 - ◆ Pro physischem Knoten genau eine Speicherseite => Effizienterer Zugriff
- Nachteil: Komplexe Reorganisation

- R-Baum



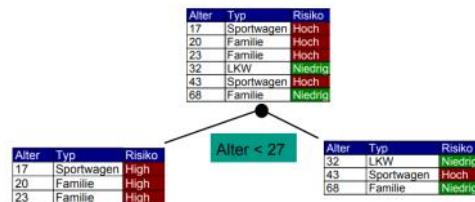
Erfasster Bildschirmausschnitt: 04.02.2018 23:07

- Prüfungsfrage: Stören uns Überlappungen der Knoten?
- Einfügen von Elementen:
 - Punkt fällt in Zone genau eines Kindknotens: unproblematisch
 - Punkt fällt in Überlappung: Optimale Entscheidung schwierig bis unmöglich
 - Punkt fällt in keine Zone eines Kundknotens:

- ◆ Einfügen in den Knoten, dessen Fläche sich am geringsten vergrößert
- ◆ Split: Zwei Kinder mit größtem Abstand finden. Achse, entlang der Abstand am größten ist, finden.
- Instanzbasiertes Lernen
 - Viele Klassifikationsverfahren existieren. Diese konstruieren in erster Phase das Modell explizit, danach Anwendung des Modells auf neue Datenobjekte.
 - Instanzbasiertes Lernen erstellt explizit kein Modell => Lazy Learning
 - Sondern: Zur Laufzeit Suche nach nächstem Nachbarn im Trainingsdatenbestand. So umgeht man Notwendigkeit zum Training
 - Benötigt Suchbaum
 - Distanzen von nominalen Attributen: Distanz = 0 gdw Attributwerte gleich, sonst 1 (oder wenn Attributwert unbekannt)
 - Kann gut mit sich ändernden Datenbeständen umgehen, jedes Attribut hat gleichen Einfluss auf Resultat (Nachteil?)
 - Probleme bei hoher Dimensionalität, schlechte Ergebnisse bei Noise.
- Schlussbemerkung: Räumliche Indexstrukturen nützlich für Clustering, Outlier Detection, Klassifikation.

Klassifikation mit Entscheidungsbäumen

- Attribute vs Features
 - Situation bei Klassifizierung/Supervised Learning:
 - Learnen eines Modells M aus Trainingsdaten, Anwendung von M auf neue Objekte
 - Trainingsdaten mit Aufbau $T=\{(x_1, y_1, l_1), (x_2, y_2, l_2), \dots\}$
 - l ist Label/Klassenzugehörigkeit, x, y, \dots sind Attribute
 - Attribute selbst nur bedingt hilfreich
 - Klassen "Motor arbeitet in fünf Minuten ohne Probleme", "Motor wird in fünf Minuten kaputt sein"
 - Absolute Temperaturen nicht aussagekräftig, mehr zB Temperaturanstieg, unterschiedliche Temperaturen in unterschiedlichen baugleichen Zylindern (letztere nennt man **Features/Merkmale**)
 - Feature: Funktion, die Attributwert auf für Klassifikation hilfreichen Wert abbildet
 - Vorgehen: Überlege alle potentiell nützlichen Features, Classifier sucht dann aus.
 - Trainingsdaten nun: $T'=\{(f_1(x_i, y_i), f_2(x_i, y_i), \dots, f_m(x_i, y_i)), \dots\}$
- Binäre Entscheidungsbäume
 - Klassifikationsansätze: Neurale Netze, genetische Algorithmen, Case-Based Reasoning, (binäre) Entscheidungsbäume
(hier: Binäre Entscheidungsbäume)
 - Baum wird aufgebaut basierend auf Training Set.
 - Unterschiedliche Entscheidungsbäume für dasselbe Trainingsdatenset möglich
 - Jedes Trainingsdatenobjekt entspricht einem Blatt
 - Datenbestand nach möglichst gut gewählten Attributen und Schwellwerten splitten (Splitattribut), um relevanten Wert aufzuteilen
Gute Wahl der Attribute und Schwellwerte



- Entropie des Splits

Entropie Wiederholung

- Definition von Entropie – Erinnerung:

$$E(S) = -\sum_j p_j \cdot \log p_j$$

p_j – relative Häufigkeit von Klasse j in S.
- Entropie ist minimal ('am wenigsten Überraschung'/ 'unser Spiel hat minimalen Pfiff'), wenn $p_i=1$; maximal, wenn $p_i=p_j$.

- Entropie eines Splits:

$$E(S_1, S_2) = \frac{n_1}{n} E(S_1) + \frac{n_2}{n} E(S_2)$$

- Ziel: Split finden, der Entropie minimiert.
- Rechenbeispiel für Splitentropie auf Folie 13,14

- Entscheidungsbau aufbauen (trivial)
 - split(Datenbestand D) { // Umgang mit Overfitting bleibt hier noch außen vor!
 - Abbruch, wenn D komplett homogen;

Split mit niedriger Entropie für D finden
Split-Attribut geeignet wählen,
Split-Wert geeignet wählen.

D-links := Menge der Tupel aus D, für die gilt:

Wert des Split-Attributs < Split-Wert

D-rechts := D - D-links;

```
split(D-links);
split(D-rechts);
}
```

- Pruning
 - Overfitting: Entscheidungsbaum ist zu sehr auf Trainingsdatenbestand zugeschnitten
 - Prepruning
 - Beim Entscheidungsbaumaufbau werden Knoten als Blatt belassen, wenn Split beliebiger Dimension nicht mehr viel bringt
 - Prepruning geschieht schrittweise. Zwei Attribute *in Kombination* bringen vielleicht noch Vorteile, kann man so nicht erkennen.
 - Postpruning
 - Entscheidungsbaum wird aufgebaut, dann zurückgeschnitten
 - IdR bevorzugt. Baum wird nur einmal aufgebaut.
 - Pruning - Abschätzung der Fehlerrate
 - Zentrale Frage: Innere Knoten durch Blatt ersetzen?
 - Fehlerrate eines Knotens abschätzen.
 - Möglichkeit: Holdout-Daten (Verfügbare Daten, die explizit nicht für Training verwendet wurden)
 - ◆ IdR bevorzugt. Aber dann kleinerer Trainingsdatenbestand.
 - Möglichkeit: Bernoulli-Experiment
 - ◆ Gute Heuristik
 - ◆ N Datenobjekte in Knothe. E Errors (#Objekte, deren Labels am häufigsten abweichen). q tatsächliche Fehlerrate (unbekannt).
 - ◆ Gemäß Bernoulli-Exoeriment Intervalle bestimmen, in dem q mit großer Wahrscheinlichkeit liegt.
 - ◆ Fehler mit Trainingsdaten immer geringer als mit Testdaten. Man nimmt obere Intervallsgrenze als Schätzwert für q.
- Wahrscheinlichkeit als Vorhersageergebnisse
 - 0 => sicher schlechter Kunde. 1 => sicher guter Kunde. 0,51 => guter Kunde, aber überhaupt nicht sicher.
 - Solche Vorhersagen von fast allen Klassifizierern möglich
 - Entscheidungsbaum: Wahrscheinlichkeit ist relative Häufigkeit der häufigsten Klasse im aktuellen Blatt.
 - Prüfungsfrage: Wie könnte es beim instanzbasierten Lernen gehen?
- Kostenberücksichtigung beim lernen
 - Fehler: False Positives, False Negatives/Misses.
 - Bankbeispiel: False Positives sind schlimmer als Misses.
 - Sensitivierung zB indem 10x mehr NO-Instanzen im Trainingsdatenbestand vorhanden sind
 - Kostenberechnung: p% eines Misses, Miss kostet k. Erwartete Kosten: p*k.

**Auf ersten Blick vielleicht unintuitiv:
Wahrscheinlichste Vorhersage i. Allg. nicht die beste!**

Beispiel (Fortsetzung des Bankbeispiels):

 - False Positive („Pleiter als Kunde“) kostet 100,
Miss (False Negative) kostet 0.
True Positive („guter Kunde richtig erkannt“) bringt 10,
True Negative („Pleiter richtig erkannt“) kostet 0.
 - Potentieller neuer Kunde: Mit 51% WS guter Kunde
gemäß unseres Classifiers, mit 49% WS schlechter Kunde.
 - Ist Vorhersage „guter Kunde“ eine gute Idee?
 - Erwarteter Gewinn R(gut' vorhergesagt | x) = 0,51 * 10 + 0,49 * (-100) = -43,9
 - Vorhersage „schlechter Kunde“ hingegen:
Erwarteter Gewinn R(schlecht' vorhergesagt | x) = 0

$P(\text{gut}' | x)$ $P(\text{schlecht}' | x)$ Cost(gut' vorhergesagt, tatsächlich ,gut')
 Cost(gut' vorhergesagt, tatsächlich ,schlecht')

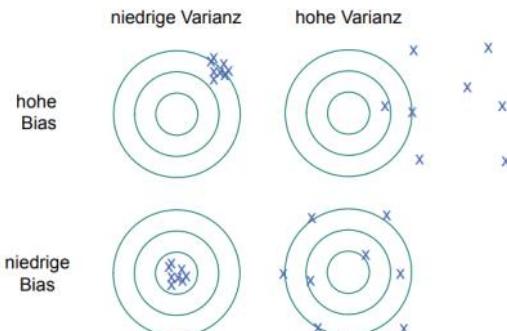
 - Conditional Risk
$$\text{Conditional Risk } R(i | x) = \sum_j P(j | x) \cdot \text{Cost}(i, j)$$
 - j – tatsächliche Klasse
 - i – vorhergesagte Klasse
- Nullwerte im Trainingsdatenbestand
 - Möglichkeit: NULL als weiteren potentiellen Wert behandeln
 - Sinnvoll, falls NULL Bedeutung impliziert: Abitur-Datum=NULL => keine höhere Bildung
 - Möglichkeit: Entscheidungsbaum, der mit gewichteten Trainingsdatenobjekten umgehen kann.
 - Bei Split-Findung werden NULL-Objekte ignoriert
 - #n₁ Objekte kommen beim Split nach links, #n₂ Objekte kommen nach rechts.
 - Objekt, das im Splitattribut Wert NULL hat, kommt mit Gewicht n₁/(n₁+n₂) nach links und mit Gewicht n₂(n₁+n₂) nach rechts.
 - => NULLObjekte anteilig ihres Gewichts berücksichtigen
- Schlussbemerkungen
 - Entscheidungsbäume beliebt wegen guter Nachvollziehbarkeit der Modelle
 - Szenarien: Trainingsdatenobjekte gleich/unterschiedlich wichtig, unterschiedliche Kosten unterschiedlicher Fehlerarten

Evaluation von Datenanalyseverfahren

- Einleitung
 - Klassifikation Vorgehen
 - Modell (Classifier) bauen, dies auf Grundlage des Trainingsdatenbestandes (kein Domänenwissen)
 - Modell für neue Vorhersagen verwenden (Prädiktion)
 - Verfügbarkeit von Trainingsdaten: Oft nicht vorhanden oder nicht so wie gewollt (zu seltene Spezialfälle)
- Training und Testing
 - Für Classifier berechenbar: Fehlerrate
 - Resubstitution Error: Fehlerrate auf Trainingsdatenbestand
 - Übliches Vorgehen: ~2/3 der Daten für Training, ~1/3 für Evaluation oÄ (Testdaten, Holdout-Daten)
 - Oft Differenzierung zwischen Trainingsdaten, Validierungsdaten, Testdaten
 - Training: Aufbau des Classifiers
 - Validierung: Parametereinstellungen, Auswahl im Fall möglicher Alternativen, usw.
 - Test: Bewertung
 - *Hinterher* Testdaten für Verfeinerung des Classifiers (wieder)verwenden
 - Crossvalidierung
 - Stratification: Sicherstellen, dass bestimmte Eigenschaften (hier: Klassenlabels) in Partitionen etwa gleich verteilt sind
 - Ziel: Fehler zuverlässig abschätzen
 - Ausgangssituation: Nur wenige Daten für Training/Testing

- Grundidee: repeated holdout Methode
 - Welcher Classifier ist für vorliegenden Datenbestand am besten?
 - Zufällige Aufteilung in Trainings- und Testdaten, mehrfach wiederholen. Jeweils Fehlerrate ermitteln, dann Durchschnitt berechnen.
 - Crossvalidierung:
 - Fixe Anzahl an Partitionen (sogenannte *Folds*). Einen Teil der Partitionen für Testen, Restliche Partitionen für Training.
 - Drei Partitionen: Drei Wiederholungen, zB Aufteilung 1/3-2/3. Dreifache Crossvalidierung (threefold cross validation)
 - 10-fold cross validation ist Standard, ebenso Stratifizierung
 - Beispiel: Daten [DDDD], prüfe mit verschiedenen Permutationen von Testdaten (P) und Trainingsdaten (T): [TTTP],[TTPT],[TPPT],[PTTT]
 - Wahrscheinlichkeiten als Vorhersageergebnisse
 - Grundprinzip: Siehe voriges Kapitel
 - Loss function: Gibt an, wieviel man durch unkorrekte Vorhersage verliert. zB 0-1 loss function: Verlust von 0 bei korrekter Vorhersage, sonst 1.
 - Quadratische Loss function
 - k mögliche Vorhersagen
 - Vorhersage liefert Vektor (p_1, \dots, p_k) mit Summe 1 (p_i ist vorhergesagte Klassenzugehörigkeit als Wahrscheinlichkeit). Tatsächliche Klassenzugehörigkeit ist Vektor (a_1, \dots, a_k) mit $a_i=1, a_{j \neq i}=0$ für ein i .
 - Quadratic Loss Function:
$$\sum_j (p_j - a_j)^2$$
 - Informational loss function
 - $-\log_2 p_i$, i ist tatsächliche Klassenzugehörigkeit
- Beispiele von eben:**
- | | |
|------------------------|--------------------------|
| ■ 1 | $-\log_2 1 = 0$ |
| ■ 0,99 | $-\log_2 0,99 = 0,0145$ |
| ■ 0,51 | $-\log_2 0,51 = 0,9714$ |
| ■ Zum Vergleich: 0,001 | $-\log_2 0,001 = 9,9658$ |
- Verfahren sollte nie Schätzung $p_i=0$ abgeben
 - Diskussion/Vergleich
 - Wie wirken sich Schätzungen für Klassen aus, die nicht zutreffend sind? Quadratic LF: restliche Wahrscheinlichkeiten sind am besten gleich groß. Informational LF: Egal.
 - Wie groß ist Verlust maximal? QLF: Nie > 2

- Fehlerarten und entsprechende Maße
 - Bias vs Varianz (Nur Begriffsbildung, geht nicht um Berechnung)



- hoher Bias: konsistenter Fehler beim Lernen
- hohe Varianz: Fehler durch Begrenztheit im Trainingsdatenbestand
- hohe Varianz => hoher zufälliger Fehler
- "Bias-Variance Decomposition"

- Fehlerarten und Erfolgsquote

		Vorhergesagte Klasse	
		yes	no
Tatsächliche Klasse	yes	true positive	false negative
	no	false positive	true negative

Mehr als zwei Klassen – sogenannte *Konfusionsmatrix*;
Beispiel:

		Vorhergesagte Klasse			
		a	b	c	total
Tatsächliche Klasse	a	88	10	2	100
	b	14	40	6	60
	c	18	10	12	40
	total	120	60	20	

- Gesamt-Erfolgsquote := $\frac{TP + TN}{TP + TN + FP + FN}$

- Gesamt-Erfolgsquote für zweite Tabelle: $(88+40+12)/200=140/200=70\%$ (Gibt also nur TP und FP)

- Kappa-Koeffizient
 - Wie gut ist diese Vorhersage (70%) wirklich?
 - Vergleich mit Classifier, der nur Anzahl Klassenzugehörigkeiten schätzt (120-60-20, unterste Zeile), ansonsten (abc x abc Block) rät
 - Beispiel:

Innerer abc x abc Block komplett geraten, selber Datenbestand wie zuvor.

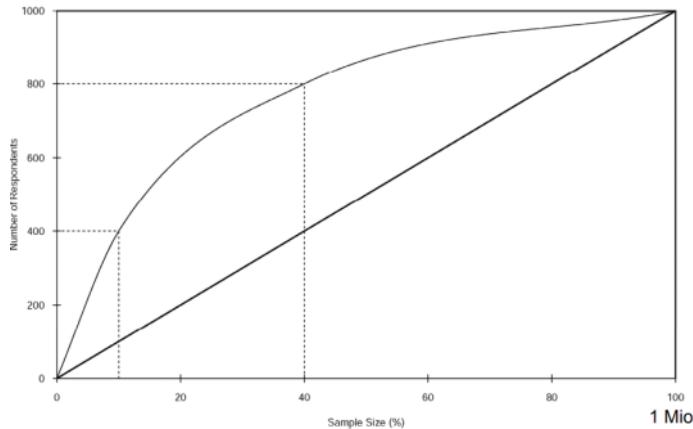
		Vorhergesagte Klasse			
		a	b	c	total
Tatsächliche Klasse	a	60	30	10	100
	b	36	18	6	60
	c	24	12	4	40
	total	120	60	20	

$$\kappa = \frac{140 - (60 + 18 + 4)}{200 - (60 + 18 + 4)} = \frac{58}{118} = 49,2\%$$

- ◆ Prüfungsfragen: Was ist Wertebereich des Kappa-Koeffizienten? Wie kommen Extremwerte zustande?
 - ◊ Classifier macht alles richtig: $K=1$
 - ◊ Classifier macht alles falsch: $K=\text{stark negativer Wert}$
 - ◊ Vorher nichts falsch: K undefiniert (?)
- Berücksichtigung der Kosten falscher Entscheidungen
 - ◆ Beispiele
 - ◊ Bank: Kreditausfall teurer, als gutem Darlehensnehmer abzusagen
 - ◊ Load Forecasting (wetterbasiert) in Energiewirtschaft: Generator unnötig hochfahren ist billiger als Kunden nicht versorgen zu können
 - ◊ Marketing: Massendrucksache an Person, die nicht antwortet vs Person nicht anschreiben, die gekauft hätte
 - ◆ Kosten von Fehlentscheidungen als Matrix darstellbar

Default Kostenmatrizen – zwei Fälle: (a) zwei Klassen, (b) drei Klassen									
		Vorhergesagte Klasse					Vorhergesagte Klasse		
		yes	no			a	b	c	
Tatsächliche Klasse	yes	0	1	Tatsächliche Klasse	a	0	1	1	
	no	1	0		b	1	0	1	
					c	1	1	0	
						(a)	(b)		

- Lift Charts als weitere Bewertungsstrategie
 - Lift Charts
 - Beispiel aus Marketing: 1mio potentielle Haushalte. Üblicher Rücklauf 0.1%, Datenanalyse identifiziert 100000 Haushalte mit Rücklauf von 0.4%, anschreiben oder nicht? Hängt von Kosten ab.
 - ◆ Lift = Faktor, um den sich Rücklaufquote erhöht. In diesem Beispiel lift=4 (Unabhängig von #identifizierteHaushälter)
 - ◆ Also: k-Haushalte anschreiben, die am ehesten antworten. Vorhergesagte Warscheinlich unwesentlich, nur Rang zählt.
 - Daten für Lift-Chart: Tabelle mit (Rank, vorhergesagt (zw 0,1), tatsächliche Klasse(ja/nein))
 - Lift-Chart



- ◆ Links oben will man sein, Diagonale bekommt man mit Raten.
- ◆ Andere Darstellung: Cost/Benefit Curve. x-Achse bleibt, y-Achse: Gewinn/Verlust, wenn Kosten eines Anschreibens und Gewinn bei Antwort quantifizierbar.

- ROC (receiver operating characteristics)

- Zugrunde liegende Definitionen:
 - FP-Rate $= 100 \times \frac{FP}{FP + TN}$
 - TP-Rate (Recall) $= 100 \times \frac{TP}{TP + FN}$

Wie oft sage ich für NO-Objekte YES vorher?

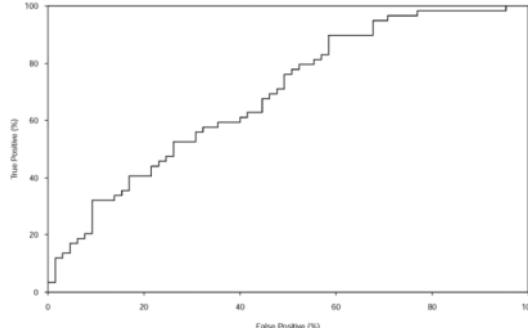
Wie oft sage ich für YES-Objekte YES vorher?

- Was ist Wertebereich der FP-Rate?
Unter welchen Umständen genau ist Wert am größten?
- Zur Erinnerung, von vorhin:

		Vorhergesagte Klasse	
		yes	no
Tatsächliche Klasse	yes	true positive	false negative
	no	false positive	true negative

ROC = „receiver operating characteristics“, Begriff aus Signalverarbeitung.

Illustration:

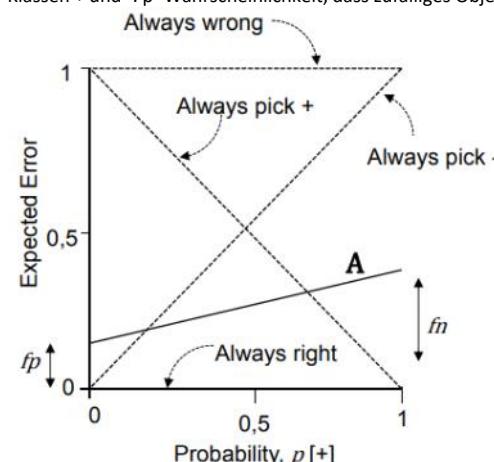


- Ausgangspunkt: Datenbestand ist nach vorhergesagter Wrscheinlichkeit sortiert
- Für erste k Datenobjekte ist bekannt: Anzahl falscher/richtiger Vorhersagen
- Links oben will man sein. Gutes Vorhersageverfahren schafft das.
- Präzision
 - x-Achse: FP-Rate (s. oben)
 - Präzision: Anteil an YES-Vorhersagen, die wirklich YES sind (\neq TP-Rate)
 - ◆ Bzw Wie oft sind YES-klassifizierte Objekte wirklich YES?
 - ◆ Precision = $100 * TP / (TP + FP)$
- ROC Zusammenfassung zu einem Wert
 - Fläche unter der Kurve
 - In etwa analog: three-point-average-recall: Durchschnittswert von 20%, 50%, 80% (oder eleven-point-average-recall: 0%, 10%, 20%, ...)
 - Weitere Kennzahlen die beide Fehlerarten berücksichtigen

$$\text{f-Measure: } \frac{2 \times \text{recall} \times \text{precision}}{\text{recall} + \text{precision}} = \frac{2 \times TP}{2 \times TP + FP + FN}$$

$$\text{Erfolgsquote: } \frac{TP + TN}{TP + FP + TN + FN}$$

- Berücksichtigung der Kosten von Fehlern
 - Klassen + und -. p=Wahrscheinlichkeit, dass zufälliges Objekt zur Klasse + gehört.



- Diagonalen entsprechen extremen Klassifizierern
- A - irgendein bestimmter Classifier
 - ◆ A-Gerade $fp * (1-p) + p * fn$ berücksichtigen
- TODO Tafelmitschrieb, mit Johanna abgleichen
- Numerische Vorhersagen - Qualitätsmaße
 - P_i ist Vorhersage für i-tes Objekt im Trainingsdatenbestand, a_i ist tatsächlicher Wert
 - Qualitätsmaße für numerische Vorhersagen

Mean-squared error	$\frac{(p_1 - a_1)^2 + \dots + (p_n - a_n)^2}{n}$
Root mean-squared error	$\sqrt{\frac{(p_1 - a_1)^2 + \dots + (p_n - a_n)^2}{n}}$
Mean-absolute error	$\frac{ p_1 - a_1 + \dots + p_n - a_n }{n}$
Relative-squared error*	$\frac{(p_1 - a_1)^2 + \dots + (p_n - a_n)^2}{(a_1 - \bar{a})^2 + \dots + (a_n - \bar{a})^2}$
Root relative-squared error*	$\sqrt{\frac{(p_1 - a_1)^2 + \dots + (p_n - a_n)^2}{(a_1 - \bar{a})^2 + \dots + (a_n - \bar{a})^2}}$
Relative-absolute error*	$\frac{ p_1 - a_1 + \dots + p_n - a_n }{ a_1 - \bar{a} + \dots + a_n - \bar{a} }$
Correlation coefficient**	$\frac{S_{PA}}{\sqrt{S_p S_A}}, \text{ mit } S_{PA} = \frac{\sum_i (p_i - \bar{p})(a_i - \bar{a})}{n-1},$ $S_p = \frac{\sum_i (p_i - \bar{p})^2}{n-1}, S_A = \frac{\sum_i (a_i - \bar{a})^2}{n-1}$

* \bar{a} ist Mittelwert des Trainingsdatenbestands.

** \bar{a} ist Mittelwert des Testdatenbestands.

□ Relative-squared Error

- ◆ Ziel: Vergleich mit einfacherem Prädiktor (Durchschnitt der Werte aus Trainingsdatenbestand)
- ◆ Zahlenbeispiel:
 - ◊ $p_1=500, a_1=600, \bar{a}_{\text{Mittelwert}}=100$
 - ◊ Quotient, wenn man nur ein Objekt hat: $100^2/500^2 = \text{kleine Zahl}$
 - ◊ $P_1=500, a_1=600, \bar{a}_{\text{Mittelwert}}=550$
 - ◊ Quotient, wenn man nur ein Objekt hat: $100^2/50^2 = \text{große Zahl}$
 - ◊ P_1 und a_1 auf unterschiedlichen Seiten vom Mittelwert. "Krasser Fehler". Zuvor: Vorhersage und tatsächlicher Wert in etwa gleich weit vom Mittelwert weg.

□ Correlation Coefficient

- ◆ Mittelwerte Testdatenbestand.
- ◆ Prüfungsfragen: Was ist maximaler Wert des Correlation Coefficients? Wann genau tritt er auf?

○ MDL - Minimum Description Length

- Erfolg beim Finden von Regelmäßigkeiten = Kürze des Modells/der Beschreibung
- Beispiel: Muster 001001001001...001 gegeben. Gutes Modell: 001*. Schlechteres Modell: 001001*
- Code: Alphabet X, Code C := injektive Abbildung von X auf {0,1}+. Lc(x) = Länge in Bits der Beschreibung (des Codes) von x
- Uniformer Code - Beispiel: X={a,b,c,d}, C(a)=00, C(b)=01, C(c)=10, C(d)=11. Lc(x)=2 for all x => Uniformer Code.
- Zusammenhang zwischen Code-Längen und Wahrscheinlichkeiten
 - $p(a)=1/2, p(b)=p(c)=1/4$. Nicht-uniformer gut gewählter Code kann kürzer kodieren als andere Codes.
 - Wie findet man optimale Länge?

Gegeben Wahrscheinlichkeitsverteilung über X.

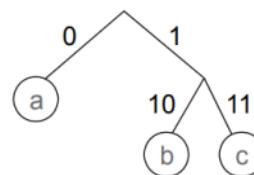
Dann gibt es Code C_P über X, für den gilt:

$$\forall x: L_{C_P}(x) = \lceil -\log P(x) \rceil$$

Es gibt Beziehung zwischen Wahrscheinlichkeitsverteilungen und Code-Längen, so dass kleine Wahrscheinlichkeiten großen Code-Längen entsprechen u. u.

Beispiel:

- $P(a) = 1/2, P(b) = P(c) = 1/4$
- $\log(1/2)$ zur Basis 2 = ?
- $\log(1/4)$ zur Basis 2 = ?
- Für alle x: $-\log P(x) = L_C(x)$
- $L_C(a) = 1, L_C(b) = L_C(c) = 2$



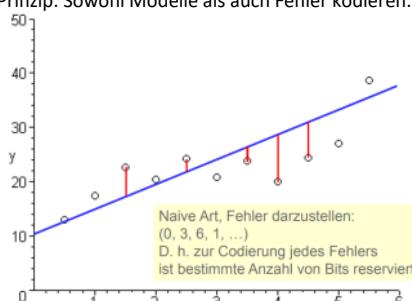
Obige Formel liefert denselben Wert für gleichverteilte Buchstaben.

Beweis in andere Richtung: Folie 68ff

- Zusammenfassung: Zusammenhang zw Wahrscheinlichkeitsverteilung über Alphabet und Code-Längen wurde thematisiert. Berechnung von CodeLängen aus WS-Verteilung und umgekehrt möglich. Minimalen Code zu verwenden entspricht MDL-Prinzip

○ MML - Minimum Message Length

- Gegeben Datenbestand $x^n = x_1, \dots, x_n$ und Modelle M_1, M_2, M_3 . Welches beschreibt den Datenbestand am besten?
- Kriterien: Fehler, Komplexität der Modelle
- Prinzip: Sowohl Modelle als auch Fehler kodieren. Modell mit kleinster GesamtCodeLänge wählen.



- Absolute Fehler nicht hilfreich, mit Wahrscheinlichkeit der unterschiedlichen Fehlerwerte (Fehler folgen bekannter Wahrscheinlichkeitsverteilung), kann man Abweichungen codieren: Kurzer Code für häufigen Fehler, langer für seltenen.
- Statistische Signifikanz (Maß an Überraschung) ist Länge des einzelnen Codes.

- Fehlerverteilung kann zB als Normalverteilung angenommen werden.
- Codierungsaufwand - Theorem (Shannon)
 - Sei x Aussage, M Modell. Wieviele Bits reichen, um x in M zu codieren? Gemäß Theorem:
 - $-\log(P(x|M))$ Bits
 - $P(x|M) = WS$, dass x eintritt, gegeben Modell M
- Zusammenfassung: Einfache Theorien (bei gleichen Inhalt) besser als umfangreiche. Vorhersagefehler als Ausnahmen der Theorie. Kompliziertes Modell nur dann, wenn es sich bezahlt macht.

MDL –Darstellung (etwas formaler)



- E – Trainingsdatenbestand
(„ E “ steht für „examples“.)
- T – Theorie,
 $L[T]$ – Anzahl Bits, um T darzustellen.
- $L[E|T]$ – Aufwand (in Anzahl Bits),
um Trainingsdatenbestand darzustellen, gegeben die Theorie.
- $L[E|T]$ – Summe der informational loss function
– Funktionswerte für alle Elemente des Trainingsdatenbestands.
- MDL empfiehlt, $L[T] + L[E|T]$ zu minimieren.

$\Pr[T|E]$ – Wahrscheinlichkeit, dass Theorie gilt, gegeben die Beispiele.

Ziel offensichtlich: T finden, so dass $\Pr[T|E]$ maximal.

$$\Pr[T|E] = \frac{\Pr[E|T] \cdot \Pr[T]}{\Pr[E]}$$

$$-\log \Pr[T|E] = -\log \Pr[E|T] - \log \Pr[T] + \log \Pr[E]$$

$\log \Pr[E]$ hängt nur vom Trainingsdatenbestand ab,
ist also unwesentlich, wenn wir unterschiedliche T s vergleichen.

D. h. gesuchte Theorie ist zugleich die, die gemäß MDL am besten ist.

- Schwierig: Wie groß ist $\Pr[T]$? D.h. wie T angemessen kodieren?

Association Rules

- Association Rules: Wichtige Art von Mustern mit einfacher Struktur. Ziel: Finde alle Association Rules, nicht Überprüfen ob eine bestimmte AR vorkommt.
 - Drücken aus, wie Phänomene zueinander in Beziehung stehen (Wenn ..., dann...)
 - Item: Einzelnes Element. Itemset: Menge von Items. Transaktion: Menge von Items, die im Datenbestand tatsächlich vorkommt.
 - Beispiel: Warenkorbkonzept
 - Items=Waren, Transaktion=Tatsächlicher Kauf (Transaktion unabhängig von DB Definition)
 - Support eines Itemsets I: Anzahl der Transaktionen, die I enthalten
 - Minimum Support σ : Schwellenwert für Support
 - Frequent Itemset: Itemset mit Support $\geq \sigma$
 - Frequent Itemsets identifizieren Mengen von Items, die positiv miteinander korrelieren, wenn der Support-Schwellenwert groß ist

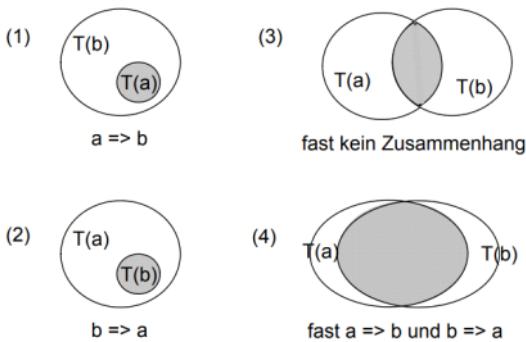
Transaction ID	Gekaufte Items
1	Milch, Obst
2	Milch, Obst, Gemüse
3	Milch
4	Obst, Brot

- $\text{Support}(\{\text{Milch}\}) = 3 (75\%)$,
 $\text{Support}(\{\text{Obst}\}) = 3 (75\%)$,
 $\text{Support}(\{\text{Milch, Obst}\}) = 2 (50\%)$.
- Wenn $\sigma = 60\%$, dann:
 $\{\text{Milch}\}$ und $\{\text{Obst}\}$ sind frequent,
aber $\{\text{Milch, Obst}\}$ ist nicht frequent.
- Frequent Itemset ist maximal gdw: es nicht Teilmenge eines anderen Frequent Itemsets ist
 - => Bestimme die maximalen FIs, dann sind alle FI bekannt (gilt nicht für AR)

- Closedness (Abgeschlossenheit)
 - Itemset ist closed, wenn es keine echte Obermenge mit genau dem gleichen Support gibt

Items a, b.

$T(a)$ – Menge der Kunden, die a kauft.



- Implikationen sind meist aussagekräftiger als "a und b werden zusammen gekauft"

- Kriterien für Auswahl der Association Rules

- $A \Rightarrow B$ [s,c]
- A und B sind Itemsets
- s = Support von $A \Rightarrow B := \text{support}(A \text{ vereinigt } B)$
 - Häufigkeit der Regel in Menge der Transaktionen. Hoher Wert: Regeln beschreibt Großteil des Datenbestandes
- c = Confidence von $A \Rightarrow B = \text{support}(A \text{ vereinigt } B) / \text{support}(A)$
 - Anteil der Transaktionen mit A, die auch B enthalten. Schätzung der bedingten WS. Wie stark ist Abhängigkeit?
 - Confidence würde nicht gebraucht, wenn nur auf Frequent Itemsets gearbeitet würde (warum?)
- Kriterien für Auswahl der Regeln: Minimum Support sigma, Minimum Confidence \gamma
 - Wir wollen nur Regeln mit $s \geq \sigma$, $c \geq \gamma$

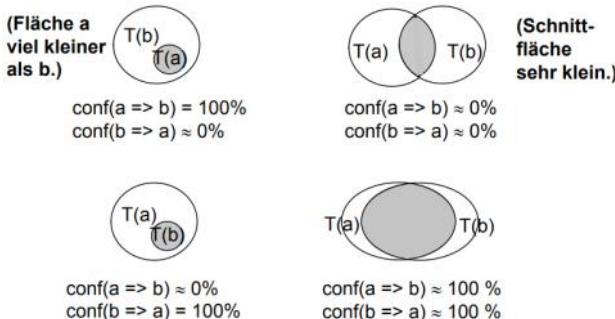
Beispiel

T1	{Zahnpasta, Schokolade, Milch}
T2	{Schokolade, Milch}
T3	{Brot, Käse}
T4	{Zahnpasta, Milch, Käse}
T5	{Milch, Brot, Käse}

	Confidence	Support
Brot \Rightarrow Käse	100%	40%
Käse \Rightarrow Milch	66.6%	40%
Zahnpasta \Rightarrow Schokolade	50%	20%

Warum hat 'Brot \Rightarrow Käse' eine andere Confidence als 'Käse \Rightarrow Brot'?

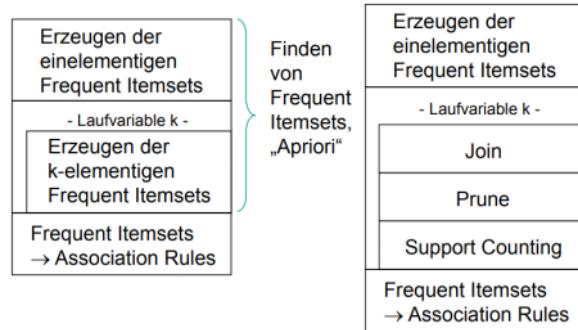
- Minimum Support \sigma
 - Hoch => wenige Frequent Itemsets. Wenige Regeln, die oft vorkommen.
 - Niedrig => Viele gültige Regeln, die selten vorkommen.
- Minimum Confidence \gamma
 - Hoch => Wenige Regeln, aber alle "logisch fast wahr"
 - Niedrig => Viele Regeln, aber viele sehr "unsicher"
- Typische Werte: $\sigma=2\text{-}10\%$, $\gamma=70\text{-}90\%$.



Beispiel für Regel mit hohem Support, aber kleiner Confidence?

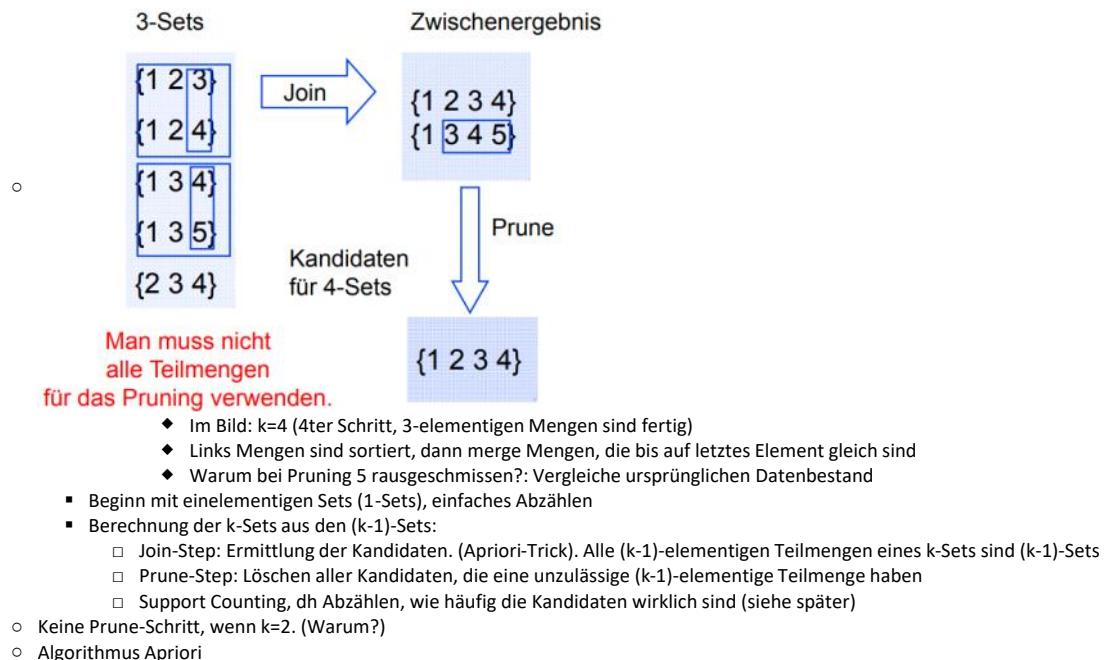
- Wenn alle Bier und Schnaps kaufen, hat „Bier \Rightarrow Schnaps“
 - hohen Support
 - und hohe Confidence.
- Wenn weniger Leute Schnaps kaufen, reduziert sich die Confidence.
- Beispiel für Association Rule mit hohem Support, aber recht geringer Confidence:
 - 30% der Kunden kaufen Bier und Schnaps.
 - 100% der Kunden kaufen Bier.
 - „Bier \Rightarrow Schnaps“ hat recht hohen Support (30%), aber eher wenig Confidence (auch 30%).

- Apriori
 - Algorithmus zum Finden von Frequent Itemsets und Association Rules



Die erste Phase ist i. Allg. die aufwendigere.

Berechnung der Kandidaten besteht aus zwei Schritten.



```

 $L_1 = \{\text{large 1-itemsets}\};$ 
for ( $k=2$ ;  $L_{k-1} \neq \emptyset$ ;  $k++$ ) do begin
     $C_k = \text{apriori-gen}(L_{k-1});$ 
        // Generierung neuer Kandidaten
        // gemäß voriger Folie
    forall transactions  $t \in D$  do begin
         $C_t = \text{subset}(C_k, t);$ 
            // candidates contained in t
        forall candidates  $c \in C_t$  do
             $c.\text{count}++;$ 
    end
     $L_k = \{c \in C_k \mid c.\text{count} \geq \text{minsup}\}$ 
end
Answer =  $\cup_k L_k;$ 

```

} Abzählen für alle Transaktionen

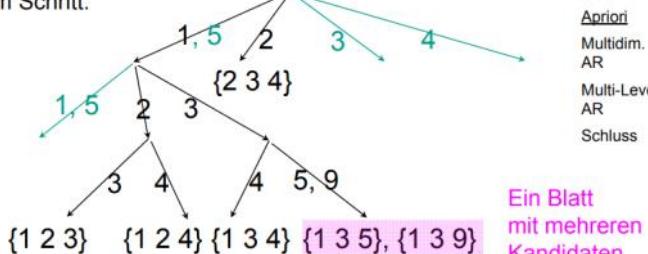
- Zum Support Counting: Ist Candidate Itemset in Transaktion t enthalten?

- Überprüfung muss effizient für viele t durchgeführt werden.
- Verwendung von zB einem Hashbaum.

Verwendung eines Hash-Trees - Beispiel:

Kandidaten: $\{1 2 3\}, \{1 2 4\}, \{1 3 4\}, \{1 3 5\}, \{1 3 9\}, \{2 3 4\}$;
Hash-Tree repräsentiert diese Menge der Kandidaten.

Hash-Tree wird einmal aufgebaut für die Kandidaten in jedem Schritt.



Einleitung
Begriffe
Assoc.
Rules
Apriori
Multidim.
AR
Multi-Lev
AR
Schluss

Ein Blatt mit mehreren Kandidaten

Wie geht also Hinzunehmen eines weiteren Itemsets?

- Beispiel: Transaktion $t=\{1,2,3,5\}$. Welche er Kandidaten ist Teilmenge von t ? Jede Transaktion wird mit Hashbaum verglichen.

Ermitteln der Association Rules aus den Itemsets

- Betrachtung aller Subsets a eines Frequent Itemsets I
- Ist $a \Rightarrow (I-a)$ eine Association Rule?

Noch einmal zur Erinnerung:

Regel $X \Rightarrow Y$ hat Support s gdw.

s % der Transaktionen X und Y enthalten;

Regel $X \Rightarrow Y$ hat Confidence c gdw. c % der Transaktionen, die X enthalten, enthalten auch Y;

(bzw. c % der Transaktionen, die X enthalten, enthalten auch $X \cup Y$)

$$\text{conf} = \frac{\text{Support}(X \cup Y)}{\text{Support}(X)}$$

$$, a \Rightarrow (I - a)' \text{ hat Confidence } \text{conf} = \frac{\text{Support}(I)}{\text{Support}(a)}$$

- ' $a \Rightarrow (I-a)'$ ist Association Rule, wenn $\text{Support}(I) \geq \text{Support}(a) * \text{minconf}$, dh. Wenn $\text{Support}(I)/\text{Support}(a) \geq \text{minconf}$
- Aufwendig, da alle Fls gebraucht werden, nicht nur maximalen

Beispiel

I={2,3,4} [40% Support]

Subsets:

	{2,3}	{2,4}	{3,4}	{2}	{3}	{4}
minconf=75%	50%	70%	60%	80%	60%	70%

$$\{2,3\} \Rightarrow \{4\}$$

$$\text{Support}(I) = 40\%$$

$$\text{Support}(a) = 50\%$$

$$\text{Confidence} = 80 \% \quad \text{OK!}$$

$$\{2\} \Rightarrow \{3,4\}$$

$$\text{Support}(I) = 40\%$$

$$\text{Support}(a) = 80\%$$

$$\text{Confidence} = 50 \% \quad \text{NO!}$$

(Da I den geforderten Support hat, haben alle Teilmengen von I ebenfalls den geforderten Support.)

- Multidimensionale Association Rules
 - Beziehungen zwischen Werten *verschiedener Attribute*
 - Beispielregeln: nationality=French=>income=high oder age=50,incomde=high=>nationality=Italian
 - Zur Verwendung von Apriori: "Abflachung" der Attribute ($<1,italian,30,low>=><1,[nat/it,age/30,income/low>$)
 - Menge von Regeln auf unterschiedlichen Hierarchieebenen.
 - Lebensmittel=>Milch,Brot,... Milch=>Diät,Voll,... Brot=>Weiß,Vollkorn...
 - Level-Crossing Association Rules: Zur Erzeugung von Kandidatenmengen können Itemsets unterschiedlicher Ebenen verknüpft werden:
 - Weißbrot=>Milch
 - Hoher Support nur mit abstrakten Konzepten. Abstrakte Regeln aber weniger interessant (zB Essware => Getränk)
 - Negativer Effekt: Menge der Frequent Itemsets wächst exponentiell mit Hierarchietiefe
 - Im folgenden vorgestellter Ansatz löst das Problem nicht, aber Beschleunigung durch integrierte Berechnung (betrachtet jede Ebene separat)
 - Also nur Multi-Level Berechnung, keine Level-Crossing Association Rules
 - Ansatz: Kodierte TA-Tabelle
 - Kodierung der Items: Milch=1, Brot=2, Diätmilch=11, Vollmilch=12, Weißbrot=22, Transaktionstabelle: {T1={11,21,22,32},T2={21,22,23}...}
 - Zweidimensionale Itemset-Tabelle L[i,j]
 - Größe der Itemsets, Tiefe der Hierarchie

Beispiel – Übersicht über die aufgebauten Tabellen

Level-1 minsup = 4

Level-1 large 1-itemsets L[1,1]	Level-1 large 2-itemsets L[1,2]	Level-1 large 3-itemsets L[1,3]			
Itemset	Support	Itemset	Support	Itemset	Support
	{1**, 2**}	4			

Level-2 minsup = 3

Level-2 large 1-itemsets L[2,1]	Level-2 large 2-itemsets L[2,2]	Level-2 large 3-itemsets L[2,3]			
Itemset	Support	Itemset	Support	Itemset	Support

Level-3 minsup = 3

...

- Ansätze zum Finden von Multi-Level Association Rules
 - Naiver Algorithmus. Beinhaltet herkömmliche Techniken zum Finden von Large Itemsets, jedes Level wird nacheinander durchlaufen.
 - Besserer Algorithmus. Erzeugung von $L[i,1]$ für alle Levels mit einem Scan
 - Wenn Item angetroffen wird, werden mehrere Zähler inkrementiert, einer für jedes Level
 - Erzeugung des k-Itemsets (für $k > 1$) wie im Single Concept Level Fall
- Apriori Diskussion: aufwendig für große Frequent Itemsets. Viele Durchläufe durch Daten, große Zwischenergebnisse.
- Zusammenfassung: Behandelte Themen: Motivation, präzise Begriffsbildung, Algorithmus, Verfeinerungen

Schnelles Bestimmen der Frequent Itemsets

- In welchen Situationen ist Apriori teuer, und warum?
- Selbes Problem: Wir suchen alle maximalen Frequent Itemsets.
- Schnelles Identifizieren von Frequent Item Sets
 - Weiterentwicklung von Apriori
 - Direct Hash and Pruning. Hash-Filter und Transaction Trimming
 - Hash-Filter
 - ◆ Beobachtung: Meist viele Kandidaten für kleine k , verglichen mit der Zahl der k -Itemsets
 - ◆ $|C_2| = (|L_1| \text{ über } 2)$
 - ◆ Hash-Filter: Beim Zählen des Supports der Elemente von C_k werden auch $(k+1)$ -elementigen Teilmengen jeder Transaktion betrachtet
 - ◆ TODO Johanna Folienmitschrieb
 - ◆ Beispiel
 - ◊ Transaktionen {1,4}, {1,4}, {1,7}, {1,7}, {2,8}, {2}, {2}, {4}, {4}, {7}, {7}, {8}, {8}
 - ◊ Minsup=2. Hashwert: Alle Werte in der Transaktion aufaddieren und mod7 berechnen.
 - ◊ Minsup=3. Hash-Funktion ist mod5. Hash-Filter offensichtlich zu grob gewählt.
 - ◆ Hash-Funktion für $(k+1)$ -Itemsets, zB h_{k+1}
 - ◆ Support Counting für alle Itemsets mit gleichem Hashwert. Übersteigen des minsups notwendig, damit alle Itemsets frequent sind.

Notation: h_2 – Hash-Funktion, H_2 – Hash-Tabelle.

Algorithmus:

```

s = a minimum support;
set all the buckets of  $H_2$  to zero;
/*  $H_2$  is hash table, i.e., array of int whose
domain is the range of the hash function */
forall transaction  $t \in D$  do begin
    insert and count 1-items occurrences
    ◆ in a hash-tree;
    forall 2-subsets  $x$  of  $t$  do
         $H_2[h_2(x)]++;$ 
end
 $L_1 = \{c \mid c.\text{count} \geq s, c \text{ is in a leaf node}$ 
 $\text{of the hash tree}\};$ 
 $c \in (L_1 * L_1)$  ist Kandidat  $\Rightarrow H_2[h_2(c)] \geq s$ .

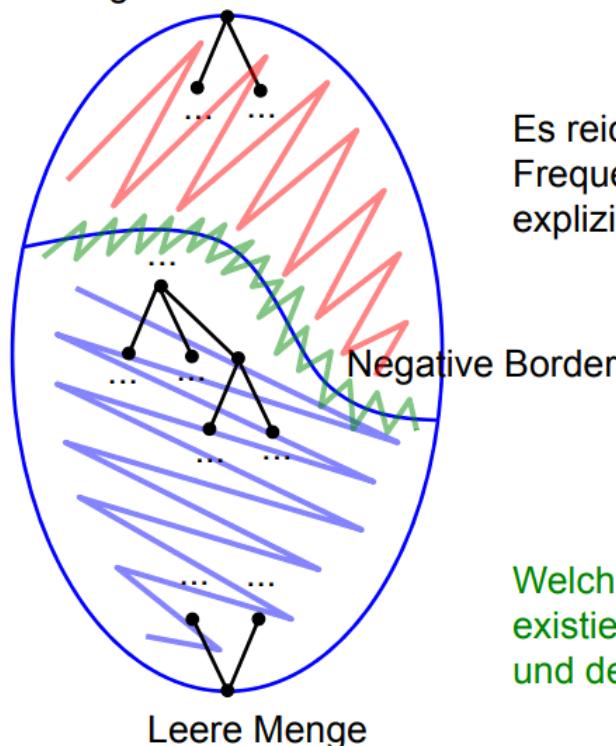
```

Wie groß sollte man Hash-Tabelle wählen?

- Prinzip: Filter basierend auf Hash-Werten. Filter kann vergleichsweise klein sein.
- Sampling
 - Ziel: Weniger Schritte über Datenbank
 - Im Bild: Punkte sind Itemsets. Kante = Nur ein Element unterschied zwischen Itemsets
 - ◆ Negative Border: Itemsets unterhalb sind frequent, die oberhalb nicht.
 - ◆ Bei größerem Support geht Linie herunter (man wird strenger)

Negative Border für Frequent Itemsets

Menge mit allen Items



Es reicht, die maximalen
Frequent Itemsets
explizit zu berechnen.

Welcher Zusammenhang
existiert zwischen minsup
und der negative border?

- Apriori erfordert gesamten DB Scan für jedes $k \Rightarrow$ hohe IO Kosten
- Ziel: Möglichst viele Berechnungen auf Stichprobe machen, die in RAM passt
- Negative Border für Sampling ermitteln. Support sowohl größer als auch kleiner wählen. Negative Border wird mit *einem* Scan über die DB geprüft.
- Sample-Größe/RAM-Größe meist groß genug, damit Negative Border stimmt \Rightarrow Ein DB-Scan reicht aus (unter Annahme, dass Sampleerzeugung billig ist)
 - ◆ Zugrunde liegende Mathematik: Bernoulli-Experiment, Elementares Ereignis, Wahrscheinlichkeitsverteilung aus großer Stichprobe berechenbar, daraus Konfidenzintervalle für Support im Gesamtdatenbestand berechenbar
- Zusammenfassung: Durchführungen von Berechnungen zumindest teilweise auf Stichprobe im RA
- Optimierung für Itemsets mit vielen Elementen | Optimist. Verfeinerungen
 - Zwei Beobachtungen bis hierhin: Alle Techniken bis jetzt erzeugen explizit alle Frequent Itemsets, nicht nur maximalen. Apriori funktioniert auch auf den Komponenten
 - Apriori-B
 - ◆ Ziel: Weitestgehend Betrachtung von nicht maximalen FI vermeiden (oben im Bild nur grüne FI betrachten, nicht blaue)
 - ◆ Idee: Frequent Itemsets in größeren Schritten durchlaufen. Insb bei großen Itemsets sinnvoll
 - ◆ Beispiel: Aus $\{1,2,3\}, \{1,2,4\}, \{1,2,5\}, \{1,2,6\}$ wird Kandidat $\{1,2,3,4,5,6\}$
- FP-Trees
 - Bisher alles Varianten von Apriori: Erzeugung von Kandidaten in jedem schritt, Scannen der DB oder eines Samples \Rightarrow Generate& Test
 - FP Trees verfolgen neues Verfahren:

- 1) Sortieren der Frequent Items innerhalb einer Transaktion nach Gesamthäufigkeit ("1 1/2 Scans" (?))

◆ [PHASE SORTIERUNG]

TID	Items	Sortierte häufige Items
100	f, a, c, d, g, i, m, p	c, f, a, m, p
200	a, b, c, f, l, m, o	c, f, a, b, m
300	b, c, h, j, o	c, b
400	b, f, k, s, p	f, b, p
500	a, f, c, e, l, p, m, n	c, f, a, m, p

- Häufigste Items in der Sortierung zuerst.
Bei gleicher Häufigkeit stets gleiche Sortierung
(hier alphabetisch).

L=<c: 4, f: 4, a: 3, b: 3, m: 3, p: 3>

- Man sieht:

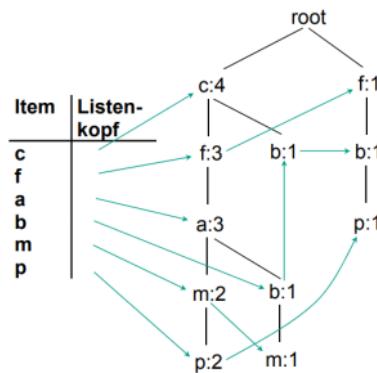
Manche Präfixe sind recht häufig, z. B. c, f, a.

- 2) Überführung der sortierten Transaktionsdatenbank in kompakte baumartige Darstellung (FP-Tree) "mit 1 1/2 Scan"

◆ [PHASE FP-TREE AUFBAUEN]

◆ FP-Tree besteht aus eigentlichem Baum und Header-Tabelle

- ◊ Baum: Jede geordnete Transaktion wird Pfad im Baum. Knoten enthält Item-ID und absolute Häufigkeit entlang des Pfades
- ◊ Header-Tabelle: Von jedem Item geht Zeiger auf verkettete Liste aus. Items werden nach Häufigkeit sortiert.



- ◊ Da häufige Präfixe wie c oder f kommen nur einmal vor

- 3) Extrahieren der Frequent Itemsets aus dem FP-Tree in Hauptspeicher

◆ [PHASE EXTRAHIEREN DER Fisets AUS FPTREE]

◆ Items werden nacheinander angefasst, beginnend mit dem am wenigsten Häufigen (1 Item pro Schritt)

- ◆ Jeder Schritt extrahiert Frequent Itemsets, die das aktuelle Item enthalten, aber keine Items, die zuvor bereits aktuell waren
- ◆ Vorgehen wie Apriori, aber ohne DB-Scan

Phase 3 – Beispiel

- Sei 3 der minimale Support.

- Item p:

- Zwei Pfade: {<(c: 4), (f: 3), (a: 3), (m: 2), (p: 2)>, <(f: 1), (b: 1), (p: 1)>},
- "Präfix-Pfade", d. h. genau die Pfade, die mit p enden: {<(c: 2), (f: 2), (a: 2), (m: 2), (p: 2)>, <(f: 1), (b: 1), (p: 1)>},
- Maximale Frequent Patterns: {fp},

- Item m:

- Zwei Pfade: {<(c: 4), (f: 3), (a: 3), (m: 2)>, <(c: 4), (f: 3), (a: 3), (b: 1), (m: 1)>},
- "Präfix-Pfade", d. h. genau die Pfade, die mit m enden, nachdem man p gelöscht hat: {<(c: 2), (f: 2), (a: 2), (m: 2)>, <(c: 1), (f: 1), (a: 1), (b: 1), (m: 1)>},
- Maximale Frequent Patterns: {cfam}

- Bei zu großem Datenbestand passt FP-Baum nicht in Hauptspeicher. Ansatz basiert auf Partitionierung der Menge der Transaktionen

- Projected Databases: Ergebnis der Partitionierung

- Ausgangspunkt: Frequent Item List L=<c:4, f:4, a:3, b:3, m:3, p:3> Transaktion, die nur Items bis einschließlich Item i enthält, kommt in i-projected Database

TID	Items	Sortierte häufige Items
100	f, a, c, d, g, i, m, p	c, f, a, m, p
200	a, b, c, f, l, m, o	c, f, a, b, m
300	b, c, h, j, o	c, b
400	b, f, k, s, p	f, b, p
500	a, f, c, e, l, p, m, n	c, f, a, m, p

- Ermitteln aller maximalen Frequent Itemsets die p enthalten (aus p-projected DataBase)

- ◆ Aufbau FP-Baum, Ermitteln dieser Fisets mit diesem Baum
- ◆ Man nimmt FP-Baum aus p-projected DB, alle Vorkommen von p herauslöschen, m-projected DB in FP-Baum einfügen.
- ◆ Vorgehen wiederholt sich, bis man alle Partitionen verarbeitet hat

- Projected DBs Erklärung

- ◆ Warum ist FPTree für Projected DB kleiner als Ausgangsdatenbestand?
- ◆ p-projected DB: Nicht alle Transaktionen enthalten p, nur wenige
- ◆ c-projected DB: Nur eine Art von Transaktion, nämlich <c> (Gar nicht erforderlich, Baum aufzubauen, Support von {c} direkt ermittelbar)
- ◆ f-projected DB: Nur Transaktionen mit c und f => Sehr kleiner FP-Baum
- ◆ 'Mittlerer' Fall: Mittlere Anzahl von Transaktionen, mittlere Anzahl von Kombinationsmöglichkeiten
- Angenommen, m-projected DB immer noch zu groß.
 - ◆ Rekursion: Partition noch feiner
 - ◆ Beispiel: Menge der Transaktionen, die p, aber nicht m enthalten. Menge der Transaktionen, die p und m enthalten.
 - ◆ Ausgangspunkt: Frequent Item List L=<...
- Zusammenfassung: FPBäume für große Datenbestände
 - Zerlegung der Datenbank in mehrere x-projected DBs. Jede x-projected DB dient dazu, bestimmte Menge von Fisets zu bestimmen.
- Diskussion: Wichtiger Vorschlag, da bis dahin alle Verfahren zur Fiset Ermittlung nur Modifikationen von Apriori waren. Sowohl Apriori als auch FP-Trees sind verallgemeinerbar für komplexere Strukturen.

Pattern Mining in Gegenwart von Constraints

- Finden häufiger Teilfolgen in Mengen von Folgen
 - Beispiel: <1,3> und <1,2,4> sind Teilfolgen von <1,2,3,4>
 - Beispiel: <1,2,3>, <1,2,4>, <4,2>, minsup=50%. <1,2> ist frequent, <2,4> nicht (also GEORDNET)
 - Prinzip wie Apriori. Kandidatenerzeugung ist Self-Join von F_{k-1}, die Join-Attribute sind die letzten k-2 Elemente der ersten und die ersten k-2 Elemente der zweiten Folge.
 - <1,2,3,4>, <2,3,4,5> -> <1,2,3,4,5>
 - Auch möglich: Listen von Mengen von Items: <{a,b},{a,c,d},{e,f,g},{a}>
- Association Rules in Gegenwart von Constraints
 - Bisher vorgestelltes Mining bietet keine Fokussierung. Für zu findene Association Rules sollen Constraints wie Geld<50€ berücksichtigt werden.
 - Ziel: Übersichtlicheres Resultat, vlt kürzere AlgoLaufzeit?
- Pruning Varianten
 - Support-basiert: Kandidat wird eliminiert (pruned), wenn er selbst oder eine seiner Teilmengen nicht frequent sind
 - Constraint-basiert: Kandidat wird eliminiert, wenn er ein aus vorgegebenen Constraints abgeleitetes Constraint nicht erfüllt
- Constraints
 - Data Constraints
 - Einschränkung auf konkrete Werte. (vgl SQL WHERE)
 - Einschränkung auf bestimmte Attribute des Raums. (vgl SQL SELECT *** FROM)
 - Rule Constraints
 - Spezifikation der Struktur oder der Eigenschaften der zu ermittelnden Regeln
 - zB Nur Fisets der Größe 3
- Ansätze (teils kombinierbar)
 - (Naiver Ansatz) Query Sprache zur Formulierung von Constraints
 - SQL-Statement als Constraint für Mining-Ergebnis
 - Nachteil: Mining-Ergebnisse müssen genau diesem Schema konform sein.
 - Mögliche weitere Ziele: Größe der Ergebnismengen nicht explizit vorgeben (hier aber der Fall), nur ein Attribut explizit vorgenommen, nicht beide.
 - Meta-Rule Guided Mining
 - Zugrundeliegende relationale DB mit Schema
 - Beispiel:
 - Zugrundeliegende Relationen:
 - student(sno, name, status, major, gpa, birth_date, birth_place, address)
 - course(cno, title, dept)
 - grading(sno, cno, instructor, semester, grade)
 - Data Mining Query/Constraint:
Finde alle Regeln der Form:
 $\text{major}(s: \text{student}, x) \wedge Q(s, y) \Rightarrow R(s, z)$
from student
where birth_place = "Canada"
in relevance to major, gpa, status, address
 - Großbuchstabe – Variable für Prädikate,
Kleinbuchstabe – Variable für entsprechenden Attributwert.
 - Constraint ist relativ rigide
hinsichtlich der Struktur der erlaubten Ergebnisse.
 - Major(s: Stud....) Heißt "Meta-Rule". Q, R sind Variablen für Prädikate, mit Attributen instantiiert, zB gpa, status, birth_place
 - Beispiel für Regeln, die gefunden werden: $\text{major}(s, "Science") \wedge \text{gpa}(s, "Excellent") \Rightarrow \text{status}(s, "Graduate")$ (60%)
 - Ziel: Sowohl Data-Constraints als auch Rule-Constraints, zB birth_place="Canada" und zwei Prädikate auf linker Seite der Regel
 - 1-var und 2-var Constraints
 - Zugrundeliegende Struktur: Menge von Items mit Attributen
 - idR Aggregation von Werten/Belegungen mehrerer aller Items
 - Beispiel: $\text{sum}(\text{LHS}) < 100 \text{ und } \text{min}(\text{LHS}) > 20 \text{ und } \text{count}(\text{LHS}) > 3 \text{ und } \text{sum}(\text{RHS}) > 1000$ (LHS = "Left-Hand Side")
 - 1-variable vs 2-variable Constraints
 - 1-var: Constraints, das nur eine Seite der Regel (L oder R) einschränkt, wie eben
 - 2-var: Constraint bezüglich beider Seiten. Beispiel: $\text{sum}(\text{LHS}) < \text{min}(\text{RHS}) \text{ und } \text{max}(\text{RHS}) < 5 * \text{sum}(\text{LHS})$
 - Hier: Regeln als Tupeln mit zwei Komponenten LHS und RHS. Verallgemeinerung mit Tuplen (S_1, \dots, S_n) mit Itemsets S_i zB Suche nach Teilstoffen, deren Elemente Itemsets sind, die nicht frequent sind.
 - Beispiele:

$$\{(S_1, S_2) \mid S_1 \subset \text{Item} \& S_2 \subset \text{Item} \& \text{count}(S_1)=1 \& \text{count}(S_2)=1 \& \text{freq}(S_1) \& \text{freq}(S_2)\}$$

– spezifiziert Frequency Constraints.

(Item-Constraints und Frequency-Constraints werden im Folgenden weggelassen.)

$$\square \{(S_1, S_2) \mid \text{agg}_1(S_1.\text{Price}) \leq 100 \& \text{agg}_2(S_2.\text{Price}) \geq 1000\}$$

- $\text{agg}_1, \text{agg}_2$ – irgendwelche Aggregationsfunktionen,
- $S_1.\text{Price}$ – Menge der Werte des Price-Attributs der Elemente von S_1 .

$$\{(S_1, S_2) \mid \text{count}(S_1.\text{Type})=1 \& \text{count}(S_2.\text{Type})=1 \& S_1.\text{Type} \neq S_2.\text{Type}\}$$

$$\square \text{Paare von Mengen von Items unterschiedlichen Typs (Type – ist hier ein Attribut.)}$$

○ 1-var Constraints

- Auflistung der Single-Variable Constraints, die im folgenden gebraucht werden:

□ Domain Constraints

- ◆ $S \theta v$. θ aus $\{=, =/=_<, <, >, >=\}$. Jedes Item aus s aus S muss Constraint $s \theta v$ erfüllen. Beispiel: $\text{Price} <= 100$
 - ◆ $v \theta S$. θ aus $\{\in, \notin\}$. Beispiel: $\text{Snack} \in \text{Type}$ (Type = Menge der Werte des Attributs Type)
 - ◆ $S \theta V, V \theta S$. θ aus $\{=, =/=_<, \subseteq, \not\subseteq, \subseteq q, \not\subseteq q\}$. Beispiel: $\{Food, Drink\} \subseteq \text{Type}$ (Also in der AR müssen Typen Food, Drink vorkommen, evtl auch andere)
- Association Rules:
- Burger, Pommes \Rightarrow Cola
 - Benzin, Frostschutzmittel \Rightarrow Schokoriegel

Zugrunde-liegende Relation	Item	Type
	Burger	Food
	Pommes	Food
	Cola	Drink
	Benzin	Non-Food
	Frostschutz	Non-Food
	Schokoriegel	Non-Food

Wenn Constraint für LHS, dann zweite Regel nicht.

Wenn Constraint für RHS, ebenfalls nicht.

(Mit Constraint ist das letzte $S \theta V$ Beispiel gemeint)

- ◆ Welche AR erfüllen $\text{Type} \in \{Food, Drink\}$? (Obige komplett, untere nur die RHS)

□ Class Constraints

- ◆ $S \in A$
- ◆ S ist Mengenvariable, A ist Attribut. S ist Menge von Werten aus Definitionsbereich von A . Beispiel: $S_1 \in \text{Item}$

□ Aggregate Constraints

- ◆ $\text{agg}(S) \theta v$. agg ist aus $\min, \max, \sum, \text{count}, \text{avg}, \theta \in \{=, =/=_<, <, >, >=\}$

○ Mining von AR unter Constraints

- Postprocessing: Naive Lösung: Finde alle Frequent Itemsets mit Apriori, überprüfe dann für jede der Mengen, ob Constraints erfüllt werden
- Optimierung: umfassende Analyse der Eigenschaften der Constraints mit Ziel, sie möglichst tief in Algorithmus "hineinzudrücken".

■ Relevante Constraint Eigenschaften:

□ Anti-Monotonität: Ziel: Constraint möglichst früh überprüfen und Pruning so früh mich möglich

- ◆ Def: Eine single-variable Constraint ist anti-monoton gdw für alle Mengen S, S' gilt:

$$S \sqsupseteq S' \& S \text{ erfüllt } C \Rightarrow S' \text{ erfüllt } C.$$

- ◆ D.h. jede Teilmenge erfüllt das Constraint.

- ◆ Beispiel: $\min(S) >= v$ ist anti-monoton.

- ◆ Prüfungsfrage: Welche Constraints sind antimonoton?: $\max(S) >= v, \text{size}(S) <= v, \text{size}(S) >= v$ [nja?]

- ◆ Warum interessant? Wenn S' Bedingung nicht erfüllt, brauchen wir mit Apriori Obermengen von S' nicht mehr betrachten

- ◊ Beispiel: Constraint $\min(S) >= 17$. $S' = \{15, 20, 25\}$

Constraint	anti-monoton	
$S \theta v, \theta \in \{=, \leq, \geq\}$	ja	, teilweise –
$v \in S$	nein	je nachdem, wie die
$S \supseteq V$	nein	Menge das Constraint
$S \subseteq V$	ja	verletzt.
$S = V$	teilweise	
$\min(S) \leq v$	nein	
$\min(S) \geq v$	ja	Beispiel für ,teilweise‘ –
$\min(S) = v$	teilweise	$\max(S) = v$. Sei $S' \subset S$.
$\max(S) \leq v$	ja	
$\max(S) \geq v$	nein	Wenn $\max(S') > v$:
$\max(S) = v$	teilweise	Man braucht S nicht
$\text{count}(S) \leq v$	ja	mehr betrachten.
$\text{count}(S) \geq v$	nein	Wenn $\max(S') < v$:
$\text{count}(S) = v$	teilweise	Das gilt nicht.
$\sum(S) \leq v$	ja	
$\sum(S) \geq v$	nein	
$\sum(S) = v$	teilweise	
$\text{avg}(S) \theta v, \theta \in \{=, \leq, \geq\}$	nein	
<frequency constraint>	<ja>	

□ Succinctness: Ziel: Kandidaten, die Constraint nicht erfüllen, werden gar nicht erzeugt

- ◆ Constraint ist succinct, wenn man alle Itemsets, die das Constraint erfüllen, explizit "in kurzer Art und Weise" hinschreiben kann

- ◆ Beispiel

- ◊ Constraint C: $S.\text{Type} = \{\text{Nonfood}\}$. Nur drei Produkte "Type=Nonfood" im Sortiment: Benzin, Frostschutz, Zewa

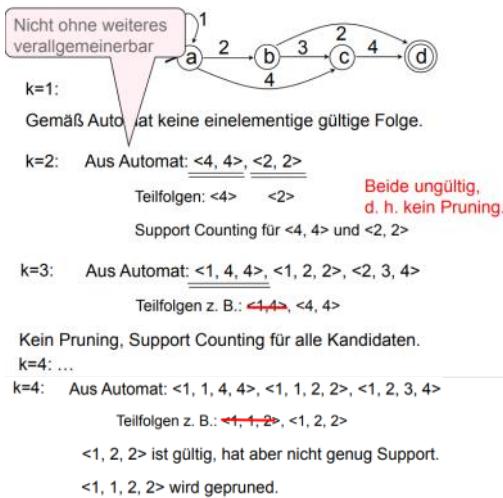
- ◊ Wieviele Frequent Itemsets gibt es maximal, die Constraint erfüllen?

- ◊ Anstelle von Apriori: Itemsets mit den drei Produkten erzeugen, ein Mal Support Counting

• Constrained Sequences

- Beispiel für nicht anti-monotones (also monotones) Constraint:
 - (ab)*. Abababab hat Teilfolge aaaa.
 - Bisheriges Vorgehen mit anti-monotonen Constraints funktioniert nicht.
- Algorithmus
 - Schritt k generiert L_k =Menge der häufigen k-Folgen, die Constraint |R erfüllen.
- Support-basiertes Pruning
 - Sei Kandidat c der Länge k gegeben. Kandidat wird eliminiert, wenn (k-1) elementige Teilfolge, die Constraint |R erfüllt, nicht frequent ist
 - Vgl Constraint-basiertes Pruning: Kandidat, der Constraint nicht erfüllt, wird eliminiert.
 - Beispiel:
 - Constraint, das nicht anti-monoton ist: (ab)*
 - Abab hat Teilfolge aab, aab ist möglicherweise frequent, aber nie in L_3
 - Beispiel
 - Constraint \mathcal{R} , abgebildet auf Automat:
 - <1,2,3,4> ist Folge, die Constraint erfüllt.
 - Diese Folge hat diverse Teilfolgen der Größe 3, z. B. <1,2,3>, <1,2,4>, <1,3,4>, <2,3,4>.
 - Aber nur eine Teilfolge erfüllt Constraint: <2,3,4>
 - Beispiel für Constraint-basiertes Pruning: <1, 2, 3> ist nicht im Ergebnis.
 - Beispiel für Support-basiertes Pruning: <1, 2, 3, 4> ist nicht im Ergebnis, wenn <2, 3, 4> $\notin L_3$.

Illustration



- Constraint-basiertes Pruning
 - Algorithmus:
 - Schritt k generiert L_k .
 - L_k – Menge der häufigen k-Folgen, die Constraint \mathcal{R} erfüllen.
 - Es scheint: Je selektiver Constraint \mathcal{R} , desto besser.
 - Nicht wirklich:
 - Angenommen, \mathcal{R} ist extrem selektiv (und nicht anti-monoton).
 - Dann funktioniert Constraint-basiertes Pruning offensichtlich gut.
 - Support-basiertes Pruning jedoch nicht!
 - ◊ Warum?: Support basiertes Pruning betrachtet alle Teilstrukturen der Größe k-1 in L_{k-1} , es gibt aber nur wenige davon.
 - Beispiel für Constraint-basiertes Pruning
 - Selber Constraint Automat. <1,2,3,4> erfüllt Constraint. Beispiele für Teilfolgen $k=3$: <1,2,3>, <1,2,4>, <1,3,4>, <2,3,4>. Nur <2,3,4> erfüllt das Constraint
 - Ang. Wahrscheinlichkeit, dass Sequence mit drei Items frequent ist, beträgt stets 50%.
 - Support-basiertes Pruning ist wirksamer ohne Constraint!
 - Weiteres Problem: Falls nicht zufällig RegExpr, ist es nicht so einfach alle Kandidaten zu erzeugen.
- Constrained vs Supportbasiertes Pruning
 - Schwächeres Constraint erleichtert Support-basiertes Pruning
 - Vielleicht Constraints zunächst abschwächen? Prüfungsfrage: Beispiel derartiger Abschwächung?
 - ◆ (ab)* könnte werden zu (a|b)*. Dann ist Constraint anti-monoton, dh Vorgehen ist klar.
 - Nicht anti-monotone Abschwächung: Gleich viele a's wie b's
 - Alternativen, wenn Constraint nicht anti-monoton:
 - Naives Postprocessing, Succinctness (nur Kandidaten generieren, die Constraint erfüllen), Kombination von Supportbasiertem und Constraintbasiertem Pruning (Hoffnung, dass Supportbasiertes besser funktioniert), Pruning mit abgeschwächtem Constraint)
- Schlussbemerkungen
 - Constrained Mining nützlich für Benutzer: Prinzip beschränkt nicht auf FI/AR, Clustering mit Einschränkungen
 - Mining beschleunigen.
 - Tradeoff Supportbasiertes Pruning vs Constraintbasiertes Pruning.

Clustering

- Allgemeine Problembeschreibung: Gegeben ist Datenmenge mit N d-dimensionalen Data Items. Finde natürliche Partitionierung der Daten in mehrere

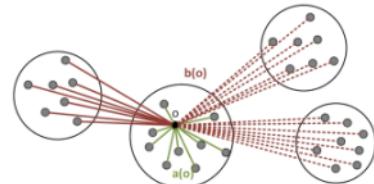
Cluster ($k = \text{Anzahl der Cluster}$) und Noise

- Cluster sollten so gewählt sein, dass Items in demselben Cluster ähnlich sind (maximale Intra-cluster Similarity) und Items in verschiedenen Clustern unterschiedlich (Inter-cluster Similarity wird minimiert).
 - Criterion Function
 - Ziel von Clustering: Criterion Function optimieren
 - Verbreitete Criterion Function für Metric Spaces:
$$E = \sum_{i=1}^k \sum_{x \in C_i} d(\bar{x}, \bar{m}_i)$$
 - Erklärungen:
 - k gegeben.
 - Minimiere E .
 - Illustration: Wenn Punkt im „falschen“ Cluster, wird E größer.
 - Was passiert, wenn k nicht gegeben ist?
 - Äußere Summe geht durch alle Cluster. m_i = Durchschnitt des i -ten Clusters, $x \setminus C_i$ sind Punkte aus Cluster
- Wahl der Parameter
 - Schwierig: Erfordert Wissen über Datenverteilung. Insb wichtig: k
 - Idee: Clusterings für $k=2, \dots, n-1$ (also für jede mögliche Anzahl an Clusters), bestes Clustering auswählen
 - Silhouette-Koeffizient
 - Maß für Qualität eines Clusterings, das nicht monoton mit k wächst (wie es für Qualitätsquantifizierung hier sein sollte)
 - Ziel: Objekte in Cluster sollten Repräsentanten des Clusters möglichst ähneln: Durchschnittlicher Abstand der Objekte zum Repräsentanten ihres Clusters. Objekte in unterschiedlichen Clustern möglichst unähnlich: durchschnittlicher Abstand von Objekt zu Objekten anderer Cluster

a(o): Durchschnittliche Distanz

zwischen Objekt o und Objekten
in seinem Cluster (A im Folgenden):

$$a(o) = \frac{1}{|C(o)|} \sum_{p \in C(o)} dist(o, p)$$



b(o): Durchschnittliche Distanz zwischen o und Objekten in „zweitnächstem“ Cluster (B im Folgenden):

$$b(o) = \min_{C_i \neq C(o)} \left(\frac{1}{|C_i|} \sum_{p \in C_i} dist(o, p) \right)$$

■ Silhouette von Objekt o: $s(o) = \begin{cases} 0 & \text{if } a(o) = 0, \text{i.e., } |C_i| = 1 \\ \frac{b(o) - a(o)}{\max\{a(o), b(o)\}} & \text{else} \end{cases}$

■ Wertebereich der Silhouette: [-1; 1]

■ Silhouette-Koeffizient eines Clusterings $C = (C_1, \dots, C_k)$:

$$silh(C) = \frac{1}{|C|} \sum_{C_i \in C} \frac{1}{|C_i|} \sum_{o \in C_i} s(o)$$

■ Wertebereich des Silhouette-Koeffizienten: [-1; 1]

■ Interpretation des Silhouette-Koeffizienten

- $s(o) = -1 \Rightarrow$ Schlecht, Objekt i ist im Mittel näher bei Elementen des nächsten Nachbarn
- $s(o) = 0 \Rightarrow$ o liegt zwischen eigenen und nächstem Cluster
- $s(o) = 1 \Rightarrow$ Gute Zuordnung von o zu seinem Cluster
- $0.7 < silh \leq 1.0 \Rightarrow$ Gute Strukturierung
- $0.5 \leq silh \leq 0.7 \Rightarrow$ Mittelmäßige Strukturierung
- $0.25 \leq silh \leq 0.5 \Rightarrow$ Schwache Strukturierung
- $silh \leq 0.25 \Rightarrow$ Keine Strukturierung

- Anforderungen: Effektive und effiziente Clustering Algorithmen für große hochdimensionale Datenbestände mit hohem Noise erfordern Skalierbarkeit bzgl #Datenelemente N, #Dimensionen d, Noiseanteil
- Distanzfunktionen

Distanzfunktion $dist(p, q)$ für einzelne Datenbankobjekte sei gegeben.

Mögliche Distanzfunktionen für Cluster
(d. h. für Mengen von Objekten):

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

- Partitionierende Verfahren

- k-Means
 - Medoid: Punkt, der als Surrogat für den Schwerpunkt dient
 - Verfahren:
 - Bestimme k Medoids p_1, \dots, p_k (Seeds) für einen gegebenen Datenbestand ("Initialisierung")
 - Ordne jeden Datenpunkt dem nächsten Medoid zu minimiere Distance Stop Criterion:
$$\sum_{i=1}^k \sum_{j=1}^{N_i} d(p_i, x_j^i)$$
 - Iterativer Algorithmus (Hill Climbing): Verschiebt jeden Medoid in Richtung des Schwerpunkts der ihm zugeordneten Menge von Punkten
 - Wähle Seeds;
 - Repeat:
 - Ordne jedes Datenobjekt dem ihm nächsten Medoiden zu, berechne Schwerpunkte dieser Partition
 - Until: Verbesserung nicht mehr signifikant
 - Initialisierungsphase: Finde möglichst gute Seeds.
 - Greedy Verfahren: Einen Seed nach dem anderen wählen
 - Eigenschaften:
 - ◆ Seeds sind selbst Datenpunkte. Neuen Seed so wählen, dass Abstand zu bisherigen Seeds groß genug ist
 - Parameter und Varianten
 - Genaues Stop-Kriterium
 - Größe der Stichprobe (Wenn Algorithmus nur auf Sample läuft)
 - Anzahl der Runs (Wenn mehrere Runs mit verschiedenen Initialisierungen gestartet werden. 5 Durchläufe reichen laut Experimenten)
 - Verfeinerungen: Medoid, dem kaum Punkte zugeordnet werden, wird ersetzt.
 - Schlechte Medoiden
 - Outlier?: Medoid des Clusters mit den wenigsten Punkten, Medoid eines Clusters mit weniger als $N/k * \text{minDev}$ Punkten
 - minDev=Konstante
 - CLARANS
 - Problem wird als Graph dargestellt. Graph nicht explizit erzeugt da sehr groß.
 - Knoten = Mögliche Auswahl an Medoiden. Menge von dafür als Medoide gewählten Objekte
 - Kanten zwischen zwei Knoten gdw ein Objekt unterschiedlich
 - CLARANS betrachtet in jedem Schritt ein Sample von Nachbar-Knoten
 - Parameter: #betrachtete Nachbarn, #Runs, Abbruchkriterium
 - Weniger unkontrolliert als voriges Verfahren, da mehrere Nachbarn betrachtet werden
 - ◆ Ersetzen eines Outliers, geschickte Wahl der Seeds gilt auch hier.
- BIRCH
 - Balanced Iterative Reducing and Clustering using Hierarchies
 - Aufbau eines Baums, der Datenverteilung beschreibt. Systematischer als partitionsbasierte Verfahren, kommt mit wenig Speicher aus.
 - Eigenschaften:
 - IO-Kosten wachsen linear mit Größe des Datenbestands, wenn CF-Tree in RAM passt
 - Eine Iteration liefert bereits ein Clustering
 - Weitere optionale Schritte liefern besseres Clustering
 - Parameter k = #gewünschter Cluster
 - Definitionen
 - Cluster $\{\vec{X}_i\}$ sei gegeben.
Cluster heißt hier allgemein „Menge von Punkten“.
 - Centroid:

$$\vec{X}_0 = \frac{\sum_{i=1}^N \vec{X}_i}{N}$$
 - Radius:

$$R = \sqrt{\frac{\sum_{i=1}^N (\vec{X}_i - \vec{X}_0)^2}{N}}$$
 Durchschnittlicher Abstand vom Mittelpunkt
 - Durchmesser:

$$D = \sqrt{\frac{\sum_{i=1}^N \sum_{j=1}^N (\vec{X}_i - \vec{X}_j)^2}{N(N-1)}}$$
 - 1, ..., N_1 – Indices der Punkte in Cluster 1,
 N_1+1, \dots, N_1+N_2 – dto. Cluster 2.
 - Durchschnittliche Inter-Cluster Distanz von Cluster 1 und 2:

$$D_2 = \sqrt{\frac{\sum_{i=1}^{N_1} \sum_{j=N_1+1}^{N_1+N_2} (\vec{X}_i - \vec{X}_j)^2}{N_1 N_2}}$$
 - Clustering Feature
 - ◆ Cluster: (Beliebige) Menge von Punkten
 - ◆ Informationen, die BIRCH zu jedem Cluster führt: Clustering Feature CF=(N, LS, SS):
 - ◊ N = Anzahl der Punkte im Cluster

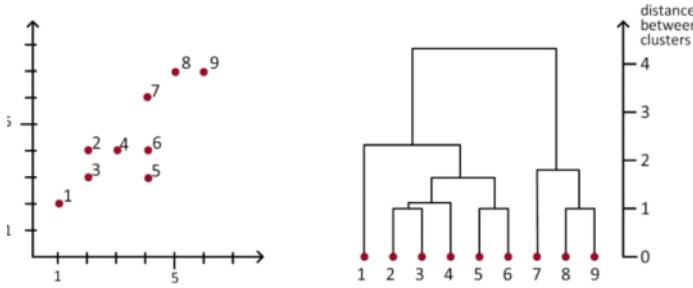
- $\overrightarrow{LS} = \sum_{i=1}^N \overrightarrow{X}_i$ (linear sum)
- $\overrightarrow{SS} = \sum_{i=1}^N (\overrightarrow{X}_i)^2$ (square sum)
 - ◆ Centroid und Radius aus CF berechenbar. (TODO: selbst versuchen)
 - ◆ Mergen von zwei Clustern: CF des neuen lässt sich trivial aus alten CFs berechnen
- CF-Tree
 - ◆ Höhenbalancierter Baum.
 - ◆ Jeder Knoten entspricht einem Cluster. Wenn Knoten zu Cluster A in Knoten zu Cluster B ist, ist A in B enthalten.
 - ◆ Blatt: Menge von Clustering Features. "Elementar-Features"
 - ◆ Innere Knoten: Liste von Einträgen [CF, child]. CF_i ist Clustering Feature von child_i.
 - ◆ Größe des Baums unabhängig von der Anzahl der Datenobjekte
 - ◆ Siehe Illustration auf Johannas Folien
 - ◆ Parameter:
 - ◊ B: Fan-Out (Kapazität für innere Knoten, Maximalzahl an Kinder für innere Knoten)
 - ◊ B': Kapazität eines Blattes (von RAM Größe abhängig)
 - ◊ T: Schwellenwert. Durchmesser oder Radius eines *Elementar*-Clusters muss kleiner als T sein.
 - ▶ Ohne T würde nur Füllgrad der Knoten die Baumstruktur steuern
 - ▶ Entferne Punkte so stets in unterschiedliche Blattknoten => Besseres Modell (Nicht alle Objekte in einem Knoten zB)
 - ◆ Einfügen in CF-Tree
 - ◊ Punkte werden in Baum eingeordnet. Punkt wandert in den Knoten, zu dessen Schwerpunkt er den kleinsten Abstand hat (Wie bei B+-Bäumen)
 - ◊ Datenobjekte werden nicht gespeichert, CF wird nur modifiziert.
 - ◊ Punkt passt in kein Cluster auf Blatt (wegen Schwellwert T)?: neuer BlattCluster
 - ◊ Innere Knoten werden gesplittet, wenn Cluster zu groß wird (zu viele TeilCluster)
 - ◊ Node-Splitting: Die am weitesten voneinander entfernten Punkte sind Seeds. Punkte werden näherem Seed zugeordnet, sobald Knoten, der einem Seed entspricht, voll ist, wird anderer Knoten aufgefüllt.
 - ▶ Nodesplitting funktioniert mit CFs, man braucht Punkte selbst nicht (Prüfungsfrage: Warum?)
 - ▶ BIRCH arbeitet nur auf aggregierten Werten, damit möglichst viel im RAM laufen kann
 - ▶ Problem: Gesplittet wird, wenn Knoten voll, unabhängig zur Qualität des aktuellen Clusterings
 - ▶ Zwischenzeitliches Zusammenfassen von Geschwister-Knoten (mergen), wenn sinnvoll
 - Platzsparen, bessere ClusteringQualität
- Knotengröße: 4

→ Große blaue Kreise sind Blätter, Kleine blaue Kreise sind Cluster. Wähle Blätter mit kleinstem Abstand. Merge diese, bemerke dass sie zu groß sind und splitte wieder.
- Knotengröße: 4

→ Einfügen in CF-Tree: Merge

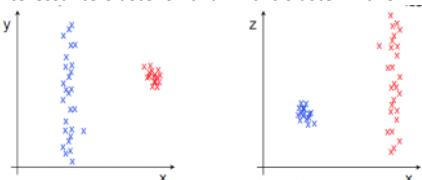
 - ▶ Merge ggf nach Splitting
 - ▶ Betrachte Kinder des Knotens, der zwei neue Knoten nach Split enthält.
 - ▶ Man fasst Kinder mit minimalem Abstand zu einem neuen Knoten zusammen (falls nicht identisch zu Split-Ergebnis)
 - ▶ Mergen erfordert ggf Resplitting der Kinder
- Clustering Algorithmus (BIRCH)?
 - ◆ Aufbau CF-Tree
 - ◆ Neuanordnung der Subcluster in den Blättern durch globalen Clustering-Algorithmus (Grund: Skewed Input, Splitting gemäß Page-Size)
 - ◆ Centroide der Cluster als Seeds nehmen, Datenpunkte neuverteilen.
- Diskussion: Idee, räumliche Indexstruktur beim Clustering zu verwenden, nicht auf partitionierende Verfahren beschränkt. Räumlicher Index ist kompakte Zusammenfassung des Datenbestandes, hier also hilfreich.
- Hierarchisches Clustering
 - Motivierendes Problem: Es gibt keine globalen Parameter für partitionierende Clustering-Verfahren zur Identifizierung aller Cluster
 - Grundlegendes Konzept
 - Hierarchische Zerlegung des Datenbestandes in Menge geschachtelter Cluster
 - Ergebnis wird als *Dendrogram* (Baum) dargestellt.
 - Knoten des Dendograms sind mögliche Cluster
 - Konstruktion bottom-up (agglomerative Ansätze) oder top-down (divisive Ansätze)

- Wurzel ist ganzer Datenbestand, Blatt ein einzelnes Datenobjekt. Innerer Knoten ist Vereinigung der Datenobjekte in seinem Teilbaum
- Höhe eines inneren Knotens: Abstand zwischen seinen zwei Kindknoten



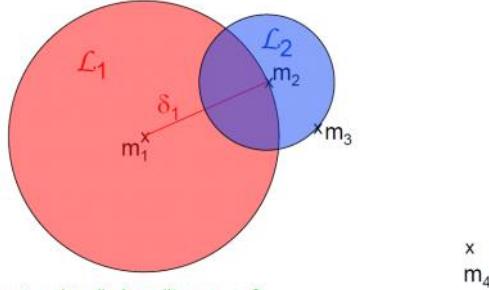
- Agglomeratives hierarchisches Clustering (bottom-up)
 - i. Jedes Objekt ist einelementiger Cluster
 - ii. Berechnung aller paarweisen Abstände zwischen diesen Clustern/zwischen Objekten
 - iii. Merge Paar {A,B} mit dem kleinsten Abstand zu neuem Cluster C:=A \ vereinigt B. Entferne A,B aus der Menge der Cluster und füge C in diese Menge ein.
 - iv. Abbruch, wenn die aktuelle Menge der Cluster nur noch aus C besteht. (Wurzelknoten)
 - v. Sonst Berechnung der Abstände von C zu allen anderen Clustern in der aktuellen Menge der Cluster. Gehe zu Schritt iii.
 - Vgl rechte Abbildung, einfacher Aufbau von unten nach oben. Prüfungsfrage: Komplexität?
- Divisives hierarchisches Clustering
 - Allgemeines Vorgehen: Ein Cluster mit allen Objekten. Wiederholung, bis allen Objekten ein einzelner Cluster entspricht.
 - Aufgaben: Auswahl eines zu splittenden Clusters (Wie?), Ersetzung des ausgewählten Clusters durch Subcluster (Wie wird gesplittet?)
 - DIANA: Beispielhafte Lösung zu "Auswahl eines zu splittenden Clusters - Wie?"
 - Auswahl des Clusters C mit größtem Durchmesser.
 - Suche nach abseitigstem Datenobjekt o in C (höchster durchschnittlicher Abstand von den anderen Objekten)
 - SplinterGroup := {o} („Splittergruppe“)
 - Wiederholtes Einfügen des $o' \in C \setminus \text{SplinterGroup}$ mit dem größten $D(o') > 0$ in SplinterGroup, bis für alle $o' \in C \setminus \text{SplinterGroup}$ gilt: $D(o') \leq 0$
 - Zweiter Summand: Durchschnittlicher Abstand zur SplinterGroup.
 - Erster Summand: dto. Nicht-SplinterGroup/Rest.
 - Wenn $D(o') \leq 0$, gehört o' eher zum Rest als zur SplinterGroup.
- Agglomeratives vs divisives hierarchisches Clustering
 - Beide benötigen $n-1$ Schritte
 - Agglomeratives betrachtet im ersten Schritt $n*(n-1)/2$ Kombinationen
 - Divisives betrachtet im ersten Schritt $2^{n-1}-2$ Möglichkeiten, den Datenbestand zu splitten. (DIANA betrachtet nicht alle Möglichkeiten explizit)
 - Divisives ist konzeptionell anspruchsvoller, weil Vorgehen beim Split nicht offensichtlich.
 - Agglomeratives berücksichtigt lokale Muster.
 - Separierendes Clustering berücksichtigt globale Datenverteilung, dadurch evtl bessere Resultate.

• Hochdimensionale Merkmalsräume

- Probleme mit hochdimensionalen Räumen
 - Interessante Cluster sind idR nicht Cluster in allen Dimensionen
 
 - Ansätze funktionieren nicht:
 - Zu geringe Dichte für Dichte-basierte Ansätze
 - Es ergibt sich keine sinnvolle Strukturierung der Daten mit hierarchischen Verfahren
 - Problem, das im Hochdimensionalen mehr Gewicht hat:
 - Dimensionen, in denen Punkte Cluster bilden, müssen erst gefunden werden.
 - Dimensionen können nicht apriori gepruned werden.
- Projected Clustering
 - Input:
 - #Cluster, die der Algorithmus finden soll = k
 - Durchschnittliche #Dimensionen pro Cluster = l (l). In obiger Grafik 2, da jeder Cluster in 2 Dimensionen verteilt ist / in zwei Richtungen den Cluster bilden
 - Output:
 - Partitionierung der Daten in $k+1$ Mengen. (Prüfungsfrage: Warum $k+1$ statt k ? TODO)
 - Teilmenge D_i der Menge der Dimensionen für jeden Cluster i.
 - Manhattan Distance:
 - Was ist der Maximalabstand im Einheitsraum? Illustration: Quadrantenlayout in Manhattan
 - Manhattan Segmental Distance: Normalisiert die Anzahl der Dimensionen heraus:

$$d_D(x_1, x_2) = (\sum_{i \in D} |x_{1,i} - x_{2,i}|) / |D|$$
 - Projected Clustering - Algorithmus
 - Wähle Seeds;
 - Repeat
 - Bestimme Dimensionen zu jedem Medoid;
 - Ordne jedes Datenobjekt dem ihm nächsten Medoid zu;
 - Berechne Schwerpunkte dieser Partition;
 - Until Verbesserung nicht mehr signifikant.
 - // Kommt dazu gegenüber zB kMeans

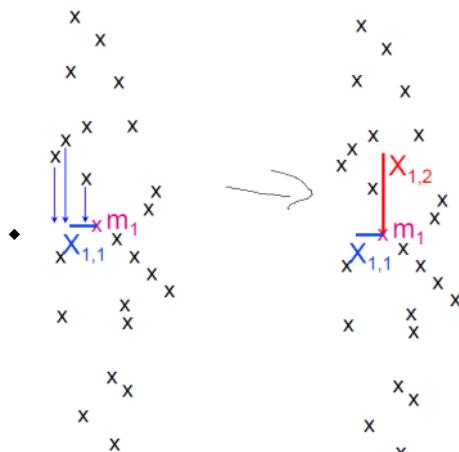
- Neuer Schritt muss jenes I-Kriterium berücksichtigen
- Ermitteln der Dimensionen zu einem Medoid
 - ◆ Unterschied zu bisher: Zu jedem Medoid wird Menge von Dimensionen explizit gewählt (in jedem Schritt neu, nachdem Medoide ermittelt)
 - Dazu für jeden Medoid m_i Punkte nahe bei m_i betrachten.
 - \mathcal{L}_i steht für diese Menge von Punkten
(„ L' steht für „Locality“/Nachbarschaft).
 - Eine mögliche Definition:
 - $\delta_i = \min_{i \neq j} (\text{dist}(m_i, m_j))$
 - Locality \mathcal{L}_i – Menge der Punkte, deren Abstand von m_i kleiner ist als δ_i .



Was wäre also die Locality von m_3 ?

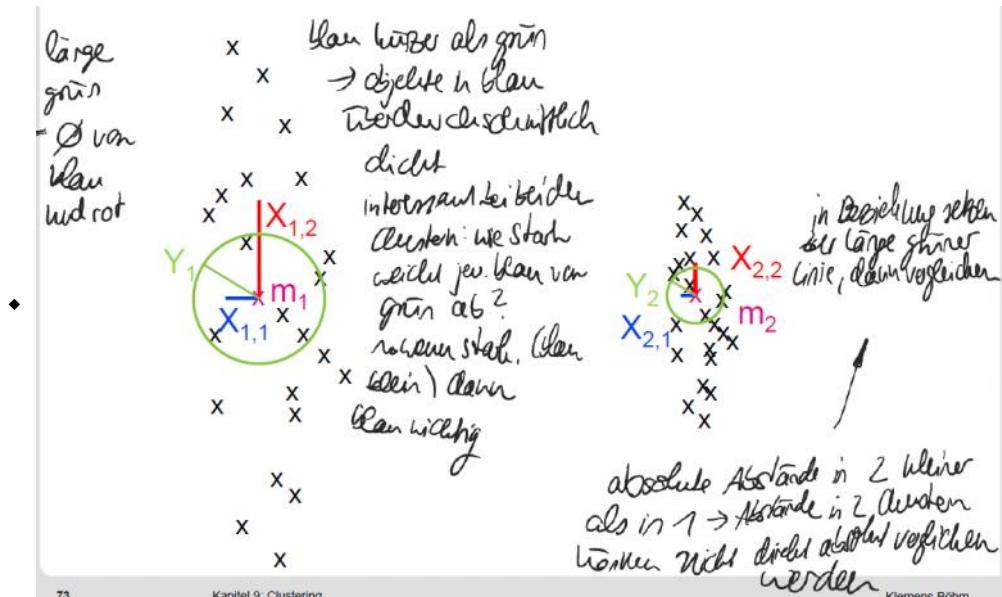
Spezialfall: Locality muss keinen weiteren Punkt enthalten, kann leer sein.

- ◆ Kreise werden so gezogen, dass der Radius genau bis zum nächsten Punkt reicht. Dann betrachte alle Punkte in dem Kreis (?) (TODO)
- ◆ Berechnung der Durchschnittsdistanz der Punkte in Lokalität_i pro Dimension j: $X_{i,j}$ (also erster Index beschreibt zugehörigen Cluster, zweiter Index beschreibt zugehörige Dimension)



- ◆ Auswahl der Dimensionen, für die die Punkte nahe bei m_i sind. Mittel der Durchschnittsdistanzen über die Dimensionen (i steht für Cluster):

$$\sum_{j=1}^d X_{i,j}$$



- ◆ Wenn $X_{i,j} - Y_i$ negativ, dann ist Dimension j wichtig für Medoid i, sonst nicht.
- ◆ Bei Betrachtung mehrerer Cluster:
 - ◊ Durchschnittszahl von Dimensionen pro Cluster i gegeben. Wie Dimensionen zuordnen?
 - ◊ Auswahl der Dimensionen, für die die Punkte nahe bei m_i sind. Verwende Y_i als Dimensionsdurchschnitt. Sind Punkte in Dimension j näher bei m_i , als im Durchschnitt? Ja, wenn $X_{i,j} - Y_i$ negativ. => Wichtig für Medoid i.
 - ◊ Interessant sind aber nicht absolute X-Werte/Differenzen, sondern relative Abweichung vom Durchschnitt.
 - ◊ Jetzt Vergleich der Differenz für unterschiedliche Cluster. Hierfür $Z_{i,j}$ berechnen:

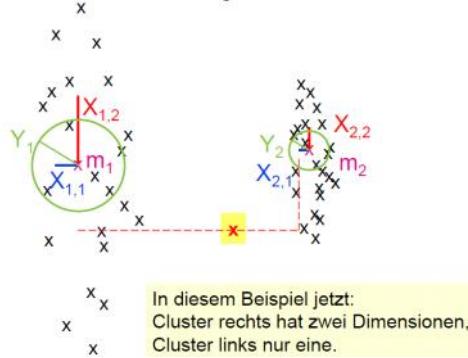
$$\diamond Z_{i,j} = \frac{X_{i,j} - Y_i}{\sigma_i}$$

Standardabweichung: $\sigma_i = \sqrt{\frac{\sum_j (X_{i,j} - Y_i)^2}{d-1}}$

- Je kleiner $Z_{i,j}$, desto eher ist Cluster i durch Dimension j beschrieben.
- Kleine Standardabweichung => Z ist groß.
- $Z_{i,j}$ sortieren für alle i und alle j und jedem Cluster damit seine Dimensionen zuordnen.

□ Bestimmung der Cluster

- ◆ Zu jedem Punkt Berechnung der Manhattan Segmental Distance zu jedem Medoid, Zuordnung des Punktes zum nächsten Medoid. Warum verwendet man Manhattan Segmental Distance? Illustration.



• Umgang mit kategorischen Attributen

- Link-basierte Methode: Datenobjekte als Punkte in Graphen darstellen, alle Punkte mit Distanz $< d$ verbinden.
 - Minimalzahl von Nachbarpunkten: 4 (lösche alle Nachbarn mit < 4 Kanten).
 - Dünn Datenbestand aus, führt zu nicht verbundene Teilgraphen die dann als Cluster interpretiert werden.
 - Prüfungsfrage: Wann könnte derartiges Vorgehen sinnvoll sein?
- Clustering für kategorische Attribute
 - Bisher nur numerische Attribute, ist aber unzureichend. (Euklidische Distanz geht auch nur auf numerischen Attributen)
 - Boolesche Attribute - Einfach 0 oder 1.
 - Kategorische Attribute - Schwieriger
 - Probleme:
 - #Items/#Attribute tendenziell groß verglichen mit Größe typischer Transaktion, dh Feature-Vektoren von sehr hoher Dimensionalität.
 - => Kunden mit ähnlichem Kaufverhalten haben wenig Items gemeinsam
 - Mengen der Items, die Cluster bestimmen, haben unterschiedliche Größen
 - ◆ Bsp: Champagner 1, ... Champagner 100, Kaviar A, ..., Kaviar Z, ..., Windeln (eine einzige Marke)
 - ◆ Also schwer Schwellenwerte zu bestimmen wenn Cluster durch Items beschrieben wird
 - Abstandsmaß ist schwer zu definieren
- Schwächen traditioneller Definitionen
 - Beispiel
 - Input-Daten {1,2,3,5},{2,3,4,5},{1,4},{6}, als Punkte: (1,1,1,0,1,0),(0,1,1,1,1,0),(1,0,0,1,0,0),(0,0,0,0,0,1) => Dann euklidischer Abstand
 - Agglomeratives Verfahren: Zunächst Merge der ersten beiden Punkte, Centroid des neuen Clusters ist (0.5, 1, 0.5, 1, 0)
 - Dann: Merge des dritten und vierten Punktes. Gemeinsamkeit: Beide kaufen sehr wenig (Cluster von Transaktionen, die keine Items gemeinsam haben) => Schlechte Wahl, Clustering berücksichtigt also nicht relevante Dinge.
 - Jaccard Koeffizient quantifiziert Ähnlichkeit von zwei Transaktionen T1 und T2:

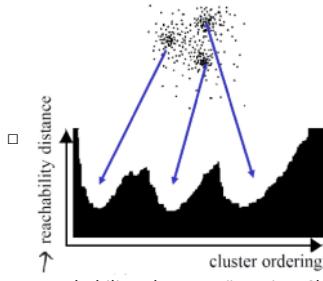
$$\frac{|T_1 \cap T_2|}{|T_1 \cup T_2|}$$
 - =1 wenn $T_1=T_2$ also komplett ähnlich, =0 wenn $T_1 \setminus T_2 = \emptyset$, also gar nicht ähnlich.
 - Beispielrechnung auf Clustering#2.17
 - {1, 2, 3}, {1, 4, 5}, {1, 2, 4}, {2, 3, 4}, {1, 2, 5}, {2, 3, 5}, {1, 3, 4},
{2, 4, 5}, {1, 3, 5}, {3, 4, 5}, {1, 2, 6}, {1, 2, 7}, {1, 6, 7}, {2, 6, 7}
 - 1,2 haben etwa alle, 3,4,5 haben die aus erstem Cluster, 6,7 haben die aus zweitem Cluster.
 - "1,2" diskriminieren nicht
 - Agglomeratives Verfahren kann scheitern, da {1,2,3} und {1,2,6} als nah empfunden werden könnte.
=> Einfach anderes auf Mengenbezogenes Abstandsmaß zu nehmen reicht nicht. Zwischen diskriminierenden und nicht-diskriminierenden Items muss unterschiedenen werden
 - ◆ Agglomerativer Algorithmus und JaccardKoeffizient vergleichen nur 2 Punkte, berücksichtigen nicht Nachbarschaft.
 - ◆ Sonst würde berücksichtigt werden, dass 2 überall vorkommt und kein Indikator für Clustering ist.

○ Lösung: Links

- Zwei Punkte sind Nachbarn, wenn Ähnlichkeit größer als Schwellenwert θ. Punkte=Transaktionen
- #Links zwischen zwei Punkten = #gemeinsamer Nachbarn dieser Punkte
- Algorithmus: Wiederholtes Merge von Clustern/Punkten mit maximaler Link-Anzahl
- Prüfungsfrage: Zeige, dass dieser Ansatz mit agglomerativem Clustering zu korrekten Clustering führt, wenn $\theta=0.5$ und Jaccard Koeffizient.
- Mögliche Veränderungen
 - Zugrundeliegendes Ähnlichkeitsmaß (statt Jaccard zB Euklid)
 - Algorithmus muss nicht agglomerativ sein
- k-Means linkbasiert
 - Datenobjekte sind Transaktionen, Seeds ebenfalls. Abstand = groß gdw wenige gemeinsame Nachbarn gemäß Jaccard.

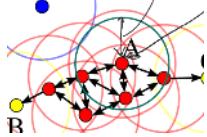
- Zusammenfassung: Clustering schwierig bei kategorischen Dimensionen. Nachbarschaft muss berücksichtigt werden. Algorithmus selbst muss dafür nicht geändert werden.
- Diskussion: Linkbasiertes Clustering geeignet für Cluster beliebiger Form/Ausdehnung, aber: Was genau wird ausgegeben? Cluster Mittelpunkt!?
- Dichte-basierte Methoden/Darstellung Cluster-Ergebnisse
 - DBSCAN
 - Objekt ist dicht (core object): min. minPts andere Objekte in Kugel um Objekt mit Radius ϵ
 - Dichte-erreichbares Objekt: Objekt in ϵ -Umgebung eines dichten Objektes, das selbst nicht dicht ist. Rand des Clusters.
 - Dichte-verbindenes Objekt: dicht oder dicht-erreichbar
 - Zuordnung dichter Punkte zu Cluster ist deterministisch, dichte-erreichbare Punkte nicht (Mehreremöglichkeiten)


```
recursiveExpandCluster(P, C, ε, MinPts)
    add P to cluster C
    if P is not visited
        mark P as visited
        N = getNeighbors(P, ε)
        if sizeof(N) >= MinPts
            for P' in N
                if P' is not yet member of any cluster
                    recursiveExpandCluster(P', C, ε, MinPts)
```
 - DBSCAN zu OPTICS
 - Statt vorgegebenem ϵ , handle alle möglichen ϵ -Werte kleiner als ϵ' gleichzeitig ab. So Datenstruktur generieren, aus der sich konkretes Clustering für beliebiges ϵ ablesen lässt. minPts ist fix.
 - Objekte so hintereinander anordnen, dass Objekte in gleichem Cluster direkt aufeinander folgen.
 - Output:

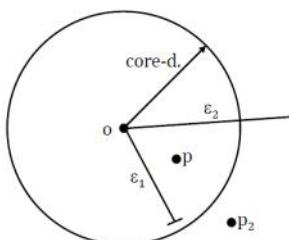


- Reachability-Plot. Repräsentiert Cluster-Struktur dichtebasierter. Leicht zu durchschauen/analysieren. Von der Dimensionalität des Datenbestandes unabhängig.

- OPTICS: Ordering Points To Identify the Clustering Structure
- Core-distance_{MinPts}(o)= Kleinstes ϵ -Wert, damit o ein dichtes Objekt ist. (grüner Kreis)



- Reachability Distance
 - Motivation: Objekte so hintereinander ordnen, dass Objekt in sortierter Liste möglichst bald auf nahe core Objekt folgt.
 - Wenn p_1 in ϵ -Umgebung von dichtem Punkt o, p_2 aber nicht, dann soll hinter o erst p_1 , dann p_2 kommen.
 - Anordnung der Objekte aus Sicht eines undichten Punktes ist dagegen uninteressant.
 - Wenn p näher an o als core Distanz,
ist genaue Distanz für die Sortierung eigentlich egal:
 - Ist ϵ kleiner als core Distanz, ist o nicht dicht.
 - Ist ϵ größer als core Distanz, ist p von o dichteerreichbar.
 - Wenn p weiter weg von o als core Distanz,
hängt es i. Allg. von ϵ ab, ob p von o dichteerreichbar.



reachability distance – Definition:

$$\text{reachDist}_{\underline{\epsilon}, \text{minPts}}(p, o) = \begin{cases} \text{dist}(p, o) & \text{if } \text{dist}(p, o) > \text{coreDist}_{\text{minPts}}(o) \\ \text{coreDist}_{\text{minPts}}(p, o) & \text{if } \text{dist}(p, o) < \text{coreDist}_{\text{minPts}}(o) \\ \text{undefined} & \text{if } \text{dist}(p, o) > \underline{\epsilon} \end{cases}$$

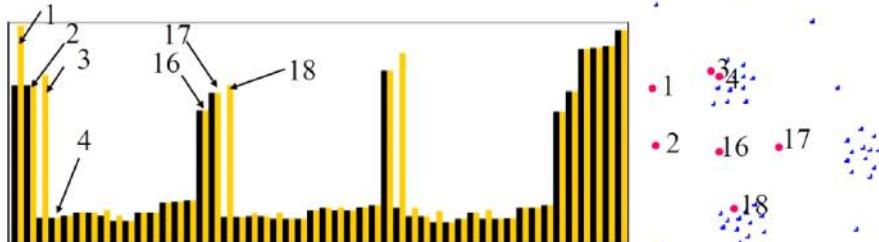
Nicht symmetrisch. core-Distanz von o geht ein, nicht aber von p.

Im Folgenden:

o wurde bereits in Ausgabeliste eingeordnet, p noch nicht.

Epsunterstrichen = $\underline{\epsilon}'$, also Obergrenze für eps

- Beispiel:



Core-distance

(minPts = 1)

wi Säulen

jeder Objekt

- 1. Zu Beginn:

1. DB enthält alle Objekte.
2. ControlList und Ausgabe – jeweils leer.

- 2. Jetzt: Objekt 1

- Objekt 1 wird ausgegeben.
- Nachbarn von Objekt 1 gemäß $\underline{\epsilon}$ in ControlList. Sortierkriterium: Abstand zu Objekt 1.

Reachability-distance

(gemeint ist: minimale reachability distance zu Objekt bereits in Ausgabeliste)

- 3. Jetzt: Erstes Objekt aus ControlList entnehmen, Objekt 2.

- Ausgabe von Objekt 2.
- Nachbarn von Objekt 2 in ControlList einfügen.
- Wenn bereits in ControlList, Abstand ggf. verkleinern.
- Neue Objekte: Abstand zu Objekt 2 als Sortierkriterium.

- 4. Jetzt: Erstes Objekt aus ControlList entnehmen, Objekt 4.

- Ausgabe von Objekt 4.
- Nachbarn von Objekt 4 in ControlList einfügen.
- Wenn bereits in ControlList, Abstand ggf. verkleinern.
- Neue Objekte: Abstand zu Objekt 4 als Sortierkriterium.

- Objekt 1: Große Coredistance, weil es zu keinem natürlichen Cluster gehört. Undefinierte Reachabilitydistance, da noch keine Bezugsobjekte.
- Objekt 2: Große coredistance, weil es zu keinem natürlichen Cluster gehört. Reachability distance ist coredistance.
- Objekt 3: Kleine coredistance wegen 2, Reachability distance ist Abstand zu 1.
- Objekt 4: Cluster von 3 bis 15.

- Algorithmus

- Beispiel: Siehe Kästen oben unter Abbildung

- Struktur wie DBSCAN, expandiere rekursiv $\underline{\epsilon}$ -Nachbarschaft, neue Objekte in ControlList einfügen. Schleife statt Rekursion.

- ControlList ist PriorityQueue mit Kriterium Minimal Reachability Distance zu irgendeinem bereits ausgegebenen Objekt. Enthält nur Objekte, die noch nicht in Output-Liste sind. Immer erst Queue leeren, bevor neues zufälliges Objekt aus DB.

- Pseudocode (nicht relevant, lieber am Beispiel oben)

```
OPTICS (SetOfObjects,  $\underline{\epsilon}$ , MinPts, OrderedFile)
  OrderedFile.open();
  FOR i FROM 1 TO SetOfObjects.size DO
    Object := SetOfObjects.get(i);
    IF NOT Object.Processed THEN
      ExpandClusterOrder(SetOfObjects, Object,  $\underline{\epsilon}$ , MinPts, OrderedFile)
  OrderedFile.close();
END; // OPTICS
```

```
ExpandClusterOrder(SetOfObjects, Object,  $\underline{\epsilon}$ , MinPts, OrderedFile);
  Object.reachability-distance := UNDEFINED;
  Object.Processed := TRUE;
  neighbors := SetOfObjects.neighbors(Object,  $\underline{\epsilon}$ );
  Object.setCoreDistance(neighbors,  $\underline{\epsilon}$ , MinPts);
  OrderedFile.write(Object);
  IF Object.core-distance <> UNDEFINED THEN {
    ControlList.update(neighbors, Object);
    WHILE NOT ControlList.empty() DO {
      currentObject := ControlList.next();
      currentObject.Processed := TRUE;
      neighbors := SetOfObjects.neighbors(currentObject,  $\underline{\epsilon}$ );
      currentObject.setCoreDistance(neighbors,  $\underline{\epsilon}$ , MinPts);
      OrderedFile.write(currentObject);
      IF currentObject.core-distance <> UNDEFINED THEN
        ControlList.update(neighbors, currentObject);}}
  END; //ExpandClusterOrder
```

```
ControlList::update(neighbors, CenterObject);
  c_dist := CenterObject.core-distance;
  FORALL Object FROM neighbors DO
    IF NOT Object.Processed THEN {
      new_r_dist := max(c_dist, CenterObject.dist(Object));
      IF Object.reachability-distance = UNDEFINED THEN {
        Object.reachability-distance := new_r_dist;
        insert(Object, new_r_dist);}}
    ELSE // Object already in ControlList
      IF new_r_dist < Object.reachability-distance THEN {
        Object.reachability-distance := new_r_dist;
        decrease(Object, new_r_dist);}}
  END; // ControlList::update
```

- Laufzeit: ca wie DBSCAN, $O(n * \text{Laufzeit}(\underline{\epsilon}\text{-Nachbarschaftsanfrage}))$

- Sensitivität der Parameter: Ziemlich insensitiv

- Gute Resultate, solange Parameter genügend groß.
- Kleineres $\underline{\epsilon}'$ => Größere Nachbarschaften bleiben außen vor (undichte große Cluster) (Aber zu groß => direkte Betrachtung des ganzen Datenbestandes)

- Kleineres MinPts => Zu kleinteilige Cluster - Objekte mit wenigen Nachbarn sind bereits dicht und bilden Cluster

- Probabilistisches Clustering

- Motivation

- Cluster durch Modelle beschreiben, zB Normalverteilung. Als Beschreibung reichen Parameter (Mittelwert, Varianz)
- Wert kann zu mehreren Clustern gehören => Finde Cluster, die am wahrscheinlichsten sind, gegeben der Daten und apriori Erwartungen
- Beispiel: Man betrachtet Kamerakäufe. Cluster "Käufe Privatpersonen", "Käufe Fotografen". 2 sich überlappende Normalverteilungen. Im Übergangsbereich lassen sich einzelne Werte nicht eindeutig zu Cluster zuordnen.
- Mixture: Menge von k (unbekannten) Wahrscheinlichkeitsverteilungen, die jeweils einen Cluster beschreiben.
 - Verteilungen haben verschiedene Parameter pro Cluster (mehr Hobbyfotografen (A) wie Profis (B)), Parameter μ_A (mittelwert), σ_{μ_A} (Standardabweichung), μ_B , σ_{μ_B} , p_A .
 - Finite-Mixture Problem:
 - ◆ Datenobjekte und #Cluster gegeben. Finde Verteilungen (dh. Parameter). Lösbar, wenn man Gauss-Verteilungen annimmt. Ohne derartige Annahme i.Allg. nicht lösbar.

- Expected Maximization

- Ang. Klassenzugehörigkeiten der Datenobjekte wären bekannt, dann kann man Parameter einfach berechnen (naive Formeln)

Umgekehrte Richtung – im Beispiel: „Wenn jemand Kamera zu bestimmtem Preis kauft, wie wahrscheinlich ist er Hobby-Fotograf?“

Formal: $A \rightarrow \text{Amateur}, x = \text{Preis}$

$$\Pr[A | x] = \frac{\Pr[x | A] \cdot \Pr[A]}{\Pr[x]} = \frac{f(x; \mu_A, \sigma_A) \cdot p_A}{\Pr[x]}$$

- Dto. $\Pr[B | x]$

$f(x; \mu, \sigma)$ Dichtefunktion

- Nenner verschwindet

wegen Normalisierung:

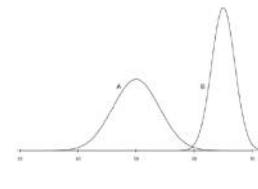
$$\Pr[A | x] + \Pr[B | x] = 1$$

- $\Pr[x | A]$ ist 0 (ebenso $\Pr[x | B]$, $\Pr[x]$),

d. h. etwas anderes als $f(x; \mu_A, \sigma_A)$.

Problem verschwindet aber

wegen Normalisierung.



- Expected Maximization (EM)-Algorithmus:

- Iteratives Vorgehen zur Lösung des Problems (FM-Problem, Finite Mixture Problem)

- 1) Initialie Parameter raten

- 2) Für jedes Datenobjekt Cluster-Wahrscheinlichkeit ausrechnen (Expectation-Schritt, voriges Bild)

- 3) Parameter der Cluster neu berechnen (Maximation-Schritt, naive Formeln verwenden)

- 4) Zurück zu 1.

- Wie sehr Datenobjekt in Berechnung der Parameter seines Clusters eingeht, hängt vom Gewicht ab:

D. h. Wahrscheinlichkeiten wirken wie Gewichte:

$$\mu_A = \frac{w_1 x_1 + w_2 x_2 + \dots + w_n x_n}{w_1 + w_2 + \dots + w_n} \quad \begin{matrix} w_1 = \text{Wk.}, \text{dann } x_1 \text{ ein Amateur} \\ \text{kauf ist} \end{matrix}$$

$$\sigma_A^2 = \frac{w_1 (x_1 - \mu)^2 + w_2 (x_2 - \mu)^2 + \dots + w_n (x_n - \mu)^2}{w_1 + w_2 + \dots + w_n}$$

- Wann terminieren? Fixpunkt nie ganz erreicht wie bei kMeans. Gütemaß des Clusterings: Overall likelihood

$$\prod_i (p_A \cdot \Pr[x_i | A] + p_B \cdot \Pr[x_i | B])$$

◆ Ang. Verteilung ist gut: Viele Instanzen nahe bei Clustermitte mu_B: $\Pr[x_i | B]$ ist groß.

◆ Ang. Clustermittelpunkte abseitig: $\Pr[x_i | A$ bzw $B]$ für viele Instanzen klein.

- Man findet nur lokales Optimum, nicht globales. => Mehrmals mit verschiedenen Initialisierungen wiederholen, Gütemaß als Auswahlkriterium.

- Erweiterung Mixture Model

- Erweiterungen

- Mehr als zwei Klassen: Trivial.

- Mehr als ein Attribut: Trivial, wenn Attribute unabhängig.

- Korrelationen zwischen Attributen.

- Kategoriale Attribute.

- Andere Verteilungen als Normalverteilungen.

▪ :

- Mehr als ein Attribut

◆ Falls unabhängig: trivial

$$\Pr[A | x, y] = \frac{\Pr[x, y | A] \cdot \Pr[A]}{\Pr[x, y]} = \frac{\Pr[x | A] \cdot \Pr[y | A] \cdot \Pr[A]}{\Pr[x, y]}$$

◆ Korrelierte Attribute

◇ Bivariate Normalverteilung (bei 2 Attributen)

◇ Statt zwei Standardabweichungen jetzt Kovarianzmatrix mit vier numerischen Einträgen (Kovarianzen zwischen Attributen)

◇ Techniken zum Schätzen der Werte.

◇ #Parameter bei n Attributen:

► Unabhängige Attribute: 2n Parameter (Mittelwert + Standardabweichung)

► Kovariante Attribute: $n + n(n-1)/2$ Parameter. N Mittelwerte und $n \times n$ Kovarianzmatrix (symmetrisch, daher $n(n-1)/2$ Parameter)

- Kategoriale Attribute

◆ Normalverteilung nicht möglich. Ang k = #Klassen, v = #Attributwerte. (Klasse = Cluster in diesem Kontext)

◆ Wie wahrscheinlich sind einzelne Werte?: $k * v$ Parameter. $\Pr[\text{konkreter Attributwert} | \text{konkrete Klasse}]$

◆ Beispiel

◇ Attribut: Kameramarke in {KA, KB, KC, KD, KE}, Klasse aus Hobby, Profi. Zehn Parameter, zB $\Pr[KC | \text{Profi}]$

◆ Expectation Schritt:

◇ Wahrscheinlichkeit der Klassenzugehörigkeit für jede Instanz berechnen.

$$\Pr[\text{Klasse } k | \text{Attributwert der Instanz}] = \frac{\Pr[\text{Attributwert} | k] \cdot \Pr[k]}{\Pr[\text{Attributwert}]}$$

◇ Beispiel: $\Pr[\text{Profi} | KC]$, rate zunächst $\Pr[\text{Attribut} | \text{Klasse}]$ und $\Pr[\text{Klasse}]$.

◆ Maximization Schritt:

◇ Attributwahrscheinlichkeiten und Wahrscheinlichkeiten ihrer Klassenzugehörigkeiten aus vorhandenen Instanzen berechnen.

◇ Bsp: x Personen kaufen KA, y kaufen KB... $\Pr[\text{Klasse } k | \text{Attributwert}]$ wurde gerade berechnet, daraus ist $\Pr[\text{Klasse } k]$ leicht berechenbar. $\Pr[\text{Attributwert} | \text{Klasse}]$ jetzt neu berechnen.

◆ $\Pr[\text{Attributwert}]$ kann 0 sein (Niemand kauft diese Marke). Dann so tun, als käme Attributwert ein Mal vor. (Laplace Estimator)

- Falls kategoriale Attribute auch korreliert:

◆ zB zwei nominale korrelierte Attribute v₁, v₂. Ersetze durch ein einzelnes Attribut mit v₁*v₂ möglichen Werten.

◆ Damit deutlich mehr Modellparameter => Overfitting.

◆ Kovariante numerische/kategoriale Attribute: Lösungen existieren, aber kompliziert.

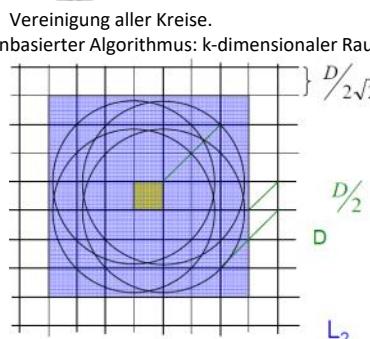
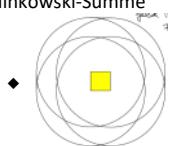
- Diskussion

- EM Algorithmus als Software verfügbar. Benutzer spezifiziert #Cluster, Typen der Attribute, explizit zu berücksichtigende Korreliertheiten, Umgang mit NULL.

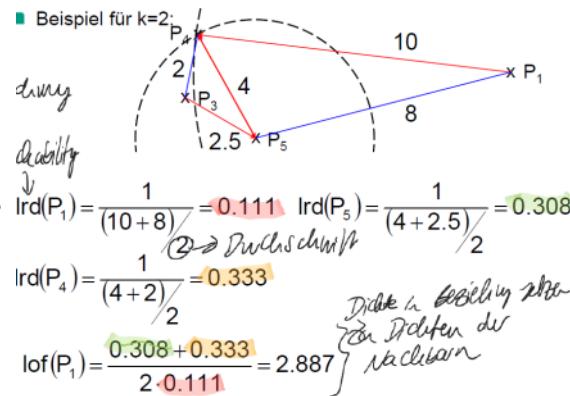
- Verteilungen: Bisher Normalverteilung, manchmal nicht angebracht.
 - ◆ Alter, Gewicht, ... haben "log"-Verteilungen: Natürliches Minimum, aber kein Maximum.
 - Log-odds Verteilungen: Attribute mit Begrenzungen "unten und oben"
 - Integer-Attributwerte: Poisson-Verteilung.
- Overfitting mit Erweiterten Mixture Modell
 - Mehr Parameter => Eher Overfitting
 - Deshalb nicht alle Attribute kovariant markieren.
 - Zu viele Cluster gewählt? => #Cluster = #Datenobjekte => Overfitting
 - Oft wird erzwungen, dass Clusters zwei Punkte brauchen.
 - Penalisierung hoher Parameterzahl erwünscht: Laplace Estimator kann das.
 - ◆ Statt Wahrscheinlichkeit = 0 für kategorische Attribute min. ein Vorkommen. Vgl overall likelihood formel. Ang Cluster A ist leer, $p_A = 0$ wird > 0 "geschummelt". => TODO
- Schlussbemerkungen
 - Clustering: Fundamental für Datenanalyse. Vielfalt an Verfahren vorgestellt.

Identifikation von Outliern

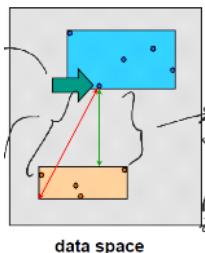
- Outlier: Element des Datenbestands, das in bestimmter Hinsicht vom restlichen Datenbestand abweicht.
- Anwendungsbeispiele: Erkennen von Kreditkartenbetrug, Videoüberwachung, Erkennen von Netzwerkfehlern, E-Commerce, Finanzen, Marketing, Data-Cleaning.
- Ansätze: Verteilungsbasiert, Clustering basiert (als Nebenprodukt vom Clustering), Abstandsbasiert, Dichtebasiert
- Verteilungsbasiert
 - Statistische Ansätze: Ang Datenbestand gemäß stat. Verteilung generiert, Test auf Auffälligkeit. Oft für Data Cleaning vor eigentlicher Datenanalyse verwendet.
 - Meisten Tests sind nur für einzelne Attribute. Verteilung oft nicht bekannt. Keine guten Ergebnisse.
- Abstandsbasiert
 - Abstandsdefinition: Objekt O im Datenbestand T ist ein DB(p, D)-Outlier, wenn der Abstand von O zu min. $p\%$ der Objekte in T größer ist als D.
 - Beispiel: 1000 Objekte, $p=99$. Dh. Höchstens 9 Objekte mit Abstand D oder weniger.
 - Algorithmen: Indexbasiert, nested-loop, Zellenbasiert
 - Indexbasiert
 - Räumliche Indexstruktur wie R-Baum
 - k-Abstand: Abstand des *k-nächsten* Nachbarn. (zB viertnächster Nachbar)
 - Wenn k-Abstand < D, dann ist Punkt kein Outlier.
 - Bei DB(p, D) ist p der Anteil der Objekte "weit weg", dh außerhalb Radius D. Wenn #Datenobjekte bekannt, k aus p berechenbar. Also einfach bestehenden Algorithmus wiederverwenden.
 - k-NN Query für jedes Datenobjekt. Stop, sobald mehr Objekte als erforderlich in Nachbarschaft.
 - Ansatz ist teuer, wenn Index erst aufgebaut werden muss.
 - Nested-Loop
 - Vgl Join Funktionsweise: Zwei Tabellen werden gejoined. Äußere Tabelle wird durchgegangen, für jeden Eintrag wird dieser festgehalten und alle Einträge der inneren Tabelle durchgegangen. Matches werden "rauskopiert". So wird "Nested Loop" mäßig iteriert in n^2 .
 - ◆ Bessere Implementierung: In äußere Tabelle werden mehrere Einträge (zB 4) festgehalten. RAM-effizient
 - Zellenbasiert
 - Minkowski-Summe
 - ◆ Vereinigung aller Kreise.
 - Zellenbasierter Algorithmus: k-dimensionaler Raum partitioniert in würfelförmige Zellen der Länge $D/2\sqrt{k}$.



- Jedem Tupel seine Zelle zuordnen. D ist der DB(p, D) definierte Maximalabstand für Nachbarn. L_2 Nachbarschaft ist der blaue Bereich, L_1 Nachbarschaft ist das unmittelbar 3x3 Raster um gelben Punkt.
 - Zellen mit weniger als m Tupeln in L_2 Nachbarschaft enthalten nur Outlier. Zellen mit mehr als m=1-p (TODO selbes m?) Tupeln in L_1 -Nachbarschaft enthalten keine Outlier, brauchen nicht betrachtet zu werden. Restliche Zellen: Outlier ermitteln, indem man Tupel einzeln inspiziert.
 - Inspektion einzelner Tupel sehr IO-lastig => nur Seiten mit Tupeln mit unklarem Status im Hauptspeicher
- k-NN Distanz: TODO Folien Outlier.25
 - Alternative: Andere Def von Outliern. Ansatz basiert auf Abstand D zw Punkt P und seinem k-NN $D_k(P)$. Ranking der Punkte basiert auf $D_k(P)$, top-n Punkte sind per Definition Outlier. TODO: War doch vorhin genau so definiert?
- Dichtebasiert
 - k-Abstand von P: Abstand des k-NN. k-Nachbarschaft $N_k(p)$: Punkte innerhalb dieses Abstands um p (inklusive k-NN)
 - Local reachability density von p: $lrd(p)$: Kehrwert des Durchschnittsabstand zwischen p und den Punkten in der k-Nachbarschaft. Abschätzung der Dichte bei p. ~> Dichte ist etwa Kehrwert des k-Abstandes. (Dichte = Anzahl Objekte in gewissem Raum)
 - Dichte allein nicht aussagekräftig genug. Illustration: Alaska Bewohner haben niedrige dichte, NYC hohe. Trump in NYC hat nie drige Dichte, interessant ist aber dass seine k-Nachbarn alle eine hohe Dichte haben.
 - Local outlier factor von p: $lof(p)$: Durchschnitt der Verhältnisse der local reachability density von p und der der k-nächsten Nachbarn von p.
 - Lof(p) ist hoch, wenn seine local reachability density niedriger als die seiner Nachbarn ist



- Algorithmus
 - Lof als Indikator für Outlier sinnvoll, aber: Berechnung des lof für alle Datenpunkte teuer. Verwende Clustering, um den Großteil der Datenpunkte zu prüfen.
 - Micro-Cluster
 - Ähnlich zu BIRCH, Verwendung von Elementarclustern. Daten werden zu Clustern komprimiert, jeder Cluster repräsentiert durch wenige aggregierte Werte wie #Datenpunkte, Mittelpunkt c und Radius r .
 - Größe der Microcluster durch maxradius begrenzt.
 - Split, indem Datenobjekte mit maximalem Abstand als Seeds genommen werden
 - Annahme: Microcluster passen in RAM
 - Mit MicroClustering kann obere und untere Schranke für lof für jeden Datenpunkt berechnet werden.



- Mögliche Untere Schranke: Grüner Pfeil. Mögliche obere Schranke: Roter Pfeil. Schranken für k -NN Abstand bzw Lof/Dichte. Verfeinerung möglich durch Betrachten der nächsten Hierarchiestufe des R-Baums
- Wenn obere Schranke des lof von P kleiner als untere Schranke des lof vieler Datenpunkte, kann P gepruned werden.
- Bis jetzt 3 Definitionen für Outlier
 - Hier verwendet: Outlier liegt in irgendeinem niedrigdimensionalen Teilraum in Region mit ungewöhnlich niedriger Dichte

- Anomalien in hochdimensionalen Datenräumen: Raum ist dünn besetzt (sparsity), hierarchische Datenstrukturen sind nicht effektiv (insb Zellen, viel zu viele Zellen die alle leer sind).

■ $n(\mathcal{D})$ – tatsächliche Anzahl Objekte in einer gegebenen Zelle.

sparsity coefficient $S(\mathcal{D}) = \frac{n(\mathcal{D}) - N \cdot f^d}{\sqrt{N \cdot f^d \cdot (1-f^d)}}$

- $S(\mathcal{D})$ negativ – Zellen, in denen Anzahl der Objekte kleiner ist als erwartet.
- Koeffizient berücksichtigt Unterschiede in den Dimensionalitäten der Teilräume.
 - Wurzel unten ist Standardabweichung ...TODO Outlier.43
- Subspace Search
 - Zahl der Teilräume, die man so betrachten kann, wächst exponentiell mit #Dimensionen. Auswahl geeigneter Teileräume, offene Forschungsfrage.

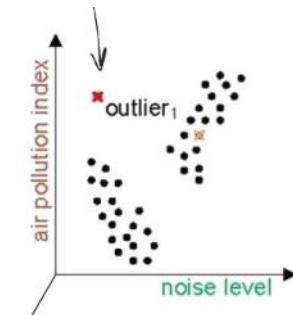
- Typen von Outliern

- P ist *nicht-trivialer Outlier im Attributraum A_p* , wenn P nicht Outlier im Teilraum $B \setminus A_p$ ist.
- Dh die kleinste Menge von Attributen, die erforderlich ist, um zu erklären, warum P besonders ist.
- P ist *strong Outlier in A_p* , wenn kein Outlier in irgendeinem $B \setminus A_p$ existiert
- Weak outlier: nicht trivial, aber auch nicht strong.
- Prüfungsfrage: Kann es einen strong outlier geben, der nicht trivial ist? (nein)

- Objektbasiert

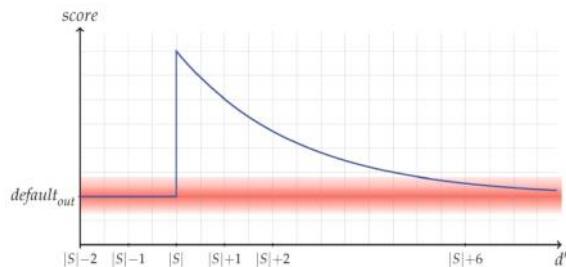
- Interessenbasiert vs Objektbasiert
 - Interessenbasiert: Raum wird nach Clustern oder Outliern abgesucht, wie in allen bisher vorgestellten Verfahren.
 - Objektbasiert: Bei riesiger Zahl an Attributen ist jedes Objekt in irgendeiner Attributkombination auffällig. Für bestimmte/alle Objekte nach genau diesen Attributen suchen. (jedoch aufwendig)
- RefOut
 - Objektbasierte Sicht im Vordergrund

- Outlier model: $\text{score}(\bar{x}_s) \in \mathbb{R} \forall \bar{x} \in \text{DB}$
- Illustration:
 - score ($x_{\text{noise level}, \text{air pollution index}}$) is large.
 - score ($x_{\text{noise level}}$) is small.
 - score ($x_{\text{noise level}, \text{air pollution index}, d_3}$) is large, but smaller than score ($x_{\text{noise level}, \text{air pollution index}}$).
 - score ($x_{\text{noise level}, \text{air pollution index}, d_3, \dots, d_{99}}$) is again small.



- Normalization
 - mean value = default_{out}
 - variance (for each subspace S) = 1

- Score = "Outlierness". Roter Punkt gutes Beispiel für nicht-trivialen Outlier: Wird eine Dimension weggelassen, fällt Outlier zu anderen Punkten ins Cluster.



- Punkt ist ab Dimension S ein Outlier. Davor gar nicht. Mit jeder weiteren Dimension wird der Outlier immer "trivialer". Es ist aber einfacher, eine Obermenge von S zu finden, als S selbst.
- Sample "Subspace Pool" wird generiert, enthält mögliche (zufällige) Teirläume (mit je in diesem Fall 9 von 12 Attributien). Für das aktuell interessante Objekt wird score (Outlierness) berechnet. Subspaces 1,2,3,4,7 zeigen: Attribute 1,2,3,4 sind interessant.

Rank	Occurrence of attributes 1-12	Outlierness
1	■■■■■■■■■■■■	■■■■■■■■■■■■
2	■■■■■■■■■■■■	■■■■■■■■■■■■
3	■■■■■■■■■■■■	■■■■■■■■■■■■
4	■■■■■■■■■■■■	■■■■■■■■■■■■
5	■■■■■■■■■■■■	■■■■■■■■■■■■
6	■■■■■■■■■■■■	■■■■■■■■■■■■
7	■■■■■■■■■■■■	■■■■■■■■■■■■
8	■■■■■■■■■■■■	■■■■■■■■■■■■
9	■■■■■■■■■■■■	■■■■■■■■■■■■
10	■■■■■■■■■■■■	■■■■■■■■■■■■
11	■■■■■■■■■■■■	■■■■■■■■■■■■
12	■■■■■■■■■■■■	■■■■■■■■■■■■
13	■■■■■■■■■■■■	■■■■■■■■■■■■
14	■■■■■■■■■■■■	■■■■■■■■■■■■
15	■■■■■■■■■■■■	■■■■■■■■■■■■
16	■■■■■■■■■■■■	■■■■■■■■■■■■
17	■■■■■■■■■■■■	■■■■■■■■■■■■
18	■■■■■■■■■■■■	■■■■■■■■■■■■
19	■■■■■■■■■■■■	■■■■■■■■■■■■
20	■■■■■■■■■■■■	■■■■■■■■■■■■

- Dann Partitionierung in $P^+_{\{1,2,3,4\}}$ (braun hinterlegt) und $P^-_{\{1,2,3,4\}}$. $S=\{1,2,3,4\}$ ist die Attributmenge, die Objekt x zu Outlier machen. T ist Attributmenge.

$$P_S^+ = \{T \mid T \supseteq S \wedge T \in P\}$$

$$P_S^- = \{T \mid S \not\subseteq T \wedge T \in P\}$$
- Suche solches S, in dem Outlierness durchschnittlich deutlich größer ist (wie im Beispiel). Menge von korrespondierenden Outliernesses zu S:

$$O_S^+ = \{\text{score}(\bar{x}_T) \mid T \in P_S^+\}$$

$$O_S^- = \{\text{score}(\bar{x}_T) \mid T \in P_S^-\}$$
- Gesucht ist Partitionierung, sodass braune Scores besser als sonstige sind. Wir kennen Durchschnitt der Samples (Durchschnitt der braunen/sonstigen Scores), wollen aber auch Erwartungswert für alle Räume kennen und diese vergleichen => Wir brauchen statistische Tests
- Score Discrepancy Problem: $E[O^+_S] > E[O^-_S]$. (TODO. In Vorlesung nicht genauer eingegangen)
- Verschiedene Outliermodels: LOF (density based), DB (distance based), FastABOD (angle based).
- Verschiedene Ansätze: Full attribute search, Random subspace selection, HiCS (?), RefOut

Ensembles

- Begriff: Wie kombiniert man verschiedene Modelle für gleichen Sachverhalt? => Ensemble = Menge solcher Modelle.
- Motivation
 - Allgemeine Lösung gegen Overfitting (nicht beschränkt auf zB Entscheidungsbäume)
 - Qualität der vorhergesagten Wahrscheinlichkeiten von Klassenzugehörigkeiten von Classifern oft sehr niedrig. Ensembles können helfen.
 - Ansätze: Bagging, Boosting, Stacking
 - Für Klassifikation; Abstimmung mit Gewichten. Numerische Vorhersagen: (gewichteter) Durchschnitt. Bei Bagging stets gleiches Gewicht, bei Boosting Berücksichtigung der Modellgüte
 - Erinnerung: Wahrscheinlichkeiten und Kosten, Conditional Risk.
- Bagging
 - Unterschiedliche Trainingsdatensamples (auf demselben Datenbestand) führen zu unterschiedlichen Classifern
 - Algorithmus

Modellerstellung:

```

Let n be the number of instances in the training data.
For each of t iterations:
    Sample n instances with replacement from training data.
    Apply the learning algorithm to the sample.
    Store the resulting model.

```

Klassifizierung:

```

For each of the t models:
    Predict class of instance using model.
Return class that has been predicted most often.

```

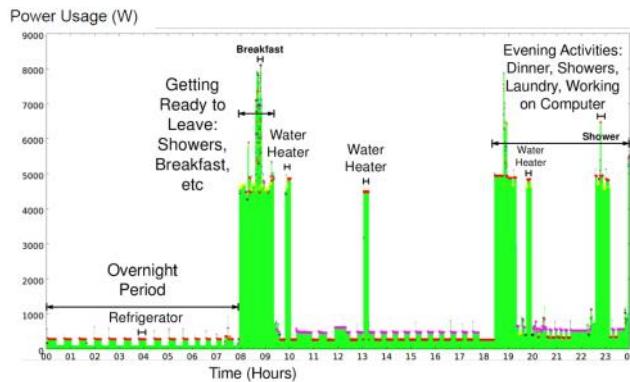
- Randomisierung
 - Mehrere Möglichkeiten, das Lernverfahren zu randomisieren (und so Ensembles konstruieren)
 - Bsp: Entscheidungsbäume: Statt Split mit größter Entropie zu wählen, aus N besten Splits einen zufälligen wählen (N nicht zu groß).
 - Bsp: NN-Classifier: Bagging bringt hier eher wenig, aber Distanzberechnung kann in unterschiedlichen, zufällig ausgewählten Teirläumen stattfinden. Verfahren bleibt unverändert.
- Diskussion:
 - Warum hilft Bagging gegen Overfitting? Resampling führt zu Datenbestandsvariationen => leicht verschiedene Classifier, lokale Besonderheiten eines Classifiers bleiben dadurch ausgeglicheniedene Class
 - Warum hilft Bagging, bessere Wahrscheinlichkeiten der Klassenzugehörigkeiten zu bestimmen? Inhalt eines Blattes ist nicht statistisch signifikant, mit Bagging: Unterschiedliche Bäume, dadurch unterschiedliche Berechnungen der Wahrscheinlichkeiten => Fehler gliechen sich aus.
- MetaCost
 - Prominentes Verfahren für Relabeling. Wendet Bagging an, um Wahrscheinlichkeiten der Klassenzugehörigkeiten mit hoher Qualität zu erzeugen. (Relabeling: Berechnung der optimalen Vorhersage im Trainingsdatenbestand, mit derartigen Labels dann trainieren)
- Boosting
 - Ziel: Objekte haben Gewicht, Classifier soll sich auf Objekte mit hohem Gewicht konzentrieren.
 - Vorgehensweise: Resampling generiert ungewichteten Datenbestand aus gewichtetem. Resampling mit Zurücklegen => Duplikate möglich. Neuer Datenbestand hat dieselbe Größe wie alter. => Am Ende sind unwichtige Objekte weniger prominent.
 - Gemeinsamkeiten mit Bagging: Voting/Durchschnittsberechnung. Modelle vom gleichen Typ, zB Entscheidungsbäume.
 - Unterschiede zu Bagging: Bei Boosting wird ein Modell nach dem anderen generiert, dabei Fokus auf bisher nicht richtig klassifizierte Objekte.
 - Forward Stagewise Additive Modeling: Neues Modell dazunehmen, das beste Gesamtperformance liefert, bisherige Modelle bleiben unverändert.
 - Modelle haben bei Vorhersage Gewicht abhängig ihrer Confidence.
 - Algorithmus AdaBoost

<p>Modellerstellung:</p> <pre> Assign equal weight to each training instance. For each of t iterations: Apply learning algorithm to weighted dataset and store resulting model. Compute error e of model on weighted dataset and store error. If e equal to zero, or e greater or equal to 0.5: Terminate model generation. For each instance in dataset: If instance classified correctly by model: Multiply weight of instance by e / (1 - e). // d.h. kleineres Gewicht für richtig klassifizierte Objekte Normalize weight of all instances. </pre>	<p>Klassifizierung:</p> <pre> Assign weight of zero to all classes. For each of the t (or less) models: Add -log(e / (1 - e)) to weight of class predicted by model. Return class with highest weight. </pre> <p>e ist Fehlerrate des jeweiligen Classifiers.</p>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

 - Zahlenbeispiele: $e=0.1 \Rightarrow e/(1-e)=0.1111$, $e=0.2 \Rightarrow e/(1-e)=0.25$, $e=0.3 \Rightarrow e/(1-e)=0.44$, $e=0.49 \Rightarrow e/(1-e)=0.96$. $-\log()$ Werte davon sind in denselben Reihenfolge: 1.996, 0.954, 0.602, 0.368, 0.017
 - Also: Kleine Fehler => korrekt klassifizierte Objekte bekommen kleines Gewicht
 - Ensemble Entscheidungen sind meist komplex und schwer nachvollziehbar. Abstraktes Vorgehen: Wie eben, Relabeling, dh Verwendung modifizierten Datenbestandes. Klassenlabels durch Ensemblevorhersage ersetzen, dafür dann ein Entscheidungsbaum
- Numerische Verfahren (IN VORLESUNG KOMPLETT ÜBERSPRUNGEN)
 - Restgröße (Residual) = Differenz zwischen vorhergesagten und tatsächlichen Werten
 - Prinzip: Konstruiere konventionelles Regressionsmodell (zB Regr Tree), konstruiere nächstes Modell für Residuals, addiere Vorhersagen der bisherigen Modelle. Ggf letzten Schritt wiederholen (Crossvalidation zum Terminierungsprüfen)
 - Summe linearer Regressionsmodelle ist wieder ein lineares RegrModell. => Erstes Modell lässt sich nicht mehr verfeinern. Lösung: Modell basierend auf einzelnen Attributen, die den Fehler minimieren (simple linear regression)
- Stacking
 - Unterschiede zu bisher: Bagging, Boosting haben zugrundeliegende Modelle vom gleichen Typ. Stacking unterstützt unterschiedliche Typen.
 - Meta-Learner kombiniert Vorhersagen
 - Level-0 Modelle: Zugrunde liegende Modelle
 - Level-1 Modelle: Metamodell. Verwendet Holotur-Set um Level-0 Modelle nachzubessern. Für Level-1 reichen einfache Verfahren, lieber "Schlichter" als wirkliche Classifier.
- Motivation: Gegen Overfitting. Bei Entscheidungsbäumen zB hilft "Zurückschneiden" des Baums. Ensembles => Allgemeineres Verfahren.
- Ensembles sind beliebt, um Classifier zu tunen. Vorhersagen meist verlässlicher als mit nur einem Classifier.

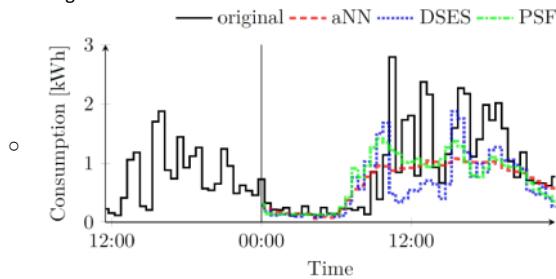
Ensembles - 2. Kapitel

- Problembeispiel: Datenströme. Folge von Werten, zB Messwerte. Ab gewissem Zeitpunkt will man Datenstrom vorhersagen.
- Versuch: Mehrere Modelle kombinieren.
- Problem, Prediction and Action zu verbinden: Prediction ist zukünftiger Aktienpreis, Aktion ist "Wann wie teuer Aktie kaufen", aber Aktion ist vom Budget, Rechtliches etc abhängig. Vlt ist Action nicht lohnenswert, nur wegen Mehraufwand.
- Anderes Beispiel: Energieverbrauch eines Hauses wird gemessen und vorhergesagt. Datenschutzproblem.



- Forschungspunkt: Daten verfälschen, um Anonymität zu schützen und trotzdem veröffentlichen können. Praktisches Beispiel: Batterie kann Stromverbrauch zu "verstecken" und auszuglätteln.

- Ausgefälscht:



- Vergleiche dann Abweichung mit zB Euklidischer Distanz, Deviation, Range coverage (jede Technik ist woanderst am Besten)

- Hard (must be met) vs Soft Requirements (Best Effort). Überlege, wie man mit Nichterfüllen umgeht.
 - Soft Requirement: zB How painful is procuring clearance for order price a_T ?
 - $f_{soft}(x_t, x_{t-1}, \dots, a_t, a_{t+1}, \dots)$. a_t ist aktuell zu fällende Entscheidung, f ist entstehender Schaden. x_i sind vergangene Messwerte, a_i vergangene tatsächliche Werte.

$$\min \sum_t f_{soft}(x_t, x_{t-1}, \dots, a_t, a_{t-1}, \dots)$$

- Summe über alle Zeitpunkte wird minimiert => Lokal beste Lösungen können suboptimal sein. Bei mehreren SoftReqs werden alle gleich relevant eingestuft.
- Aktion ist optimal, wenn alle harten Reqs erfüllt sind und es keine andere Aktion gibt, die auch alle harten Reqs erfüllt und die SoftReqs besser erfüllt.

- Verwende Ensembles zur Lösung.

- Ensembles: Menge von Modellen, deren Vorhersagen zusammengeführt werden, um finale Vorhersage zu treffen. Ensembles können gewichtet werden.
- Gut, wenn unbekannt welcher Prediktor gut ist. Aber: Qualitätsmaß notwendig um Gewichte zu lernen. Auch Problem: Manche Ensembles brauchen einen Prädiktor pro Ensemble => Informationsverlust (TODO?)
- Lösung

- Suche nicht explizit nach gutem Qualitätsmaß, versuche Vorhersage direkt zu Entscheidungsqualität zuzuordnen. Behandle Suche nach guter Entscheidung als Optimierungsproblem.

- HardReqs: Lösungssuchraum. SoftReqs: Qualitätswerte verschiedenen Lösungen zuweisen.
- Verwende bestehende Optimierungsansätze

- Ensemble Phasen: Ensemblemembers mit Trainingsdaten trainieren => Gewichte lernen (mit anderen Trainingsdaten) => Deploy Ensemble (on new data). Herkömmlicher Prädiktor: Train => Deploy.

- Framework

- Input: f_{strict} , f_{soft} , Vorhersagetechniken $m \in [1, M]$, Optimierungsmethode (für Phase 3), Datenströme x_1, \dots, x_{ω} , Zeitpunkte t_1, t_2
- Output: Aktionen $a_{t+1}, \dots, a_{\omega}$

- Vorgehensweise:

- ◆ Vorhersagetechniken m werden auf Datenströmen x_1, \dots, x_{t_1} trainiert.

- Jeder Prädiktor wird isoliert trainiert.

- ◆ Gewichte w_1, \dots, w_M für Vorhersagetechniken werden auf wrt Anforderungen (?) x_{t+1}, \dots, x_{t_2} entschieden.

- Einzelne Ensemble Mitglieder gewichten, um deren Ergebnisse zu bewerten

$$quality(t) = \begin{cases} -\sum_{f_{soft}}^{+\infty} f_{soft}(x_t, x_{t-1}, \dots, a_t, a_{t-1}, \dots) & \text{if a strict constraint is violated} \\ else \end{cases}$$

- Pro Vorhersagetechnik pro Zeitpunkt. Kein Wiederverwenden von Daten aus Phase 1 => Kein Overfitting.

- Input: Original Records x_t und vorhergesagte Records x_t^m für t aus $[t_1+1, t_2]$ und m aus $[1, M]$, sowie f_{strict} und f_{soft}

- Output: Gewichte w_m für m in $[1, M]$

- Vorgehensweise

- Determine actions a_t^m for predictions x_t^m

- the ones that minimize the soft constraints.

- Determine actions a_t^{opt} for records x_t in the same way

- For $t \in [t_1 + 1, t_2]$ do

↳
$$quality(opt, t) = - \sum_{f_{soft}} f_{soft}(x_t, x_{t-1}, \dots, a_t^{opt}, a_{t-1}^{opt}, \dots)$$

- For $m \in [1, M]$ do

↳
$$quality(m, t) = - \sum_{f_{soft}} f_{soft}(x_t, x_{t-1}, \dots, a_t^m, a_{t-1}^m, \dots)$$

↳
$$diff(m, t) = (quality(opt, t) - quality(m, t))^2$$

- Normalize diff and convert to weights.

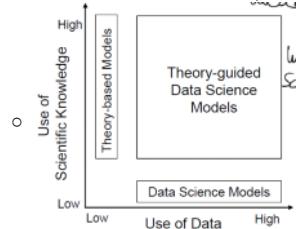
- Weight determination – similar to the one from Boosting
(an established ensemble technique).

- ◆ Bestimme optimale Entscheidungen $a_{t+1}, \dots, a_{\omega}$, wrt-Anforderungen, Vorhersagen und deren Gewichte, gebe $a_{t+1}, \dots, a_{\omega}$ aus.

- t_1 ist Dauer der Trainingsphase, t_2 ist Dauer der Gewichtungsphase.
- Schwache Techniken (zB "Nehme Wert letzter Woche") sind erlaubt, geringes Gewicht, kann so für exotische Anforderungen helfen.
- Experimente
 - Vorhersagetechniken: Simple/Double Exponential Smoothing, Pattern Sequence Based Forecasting, Artificial Neural Network, Floating Average, "Same value as previous day/week", Aggregation
 - Optimisierung: Nelder-mead Methode.
 - StrictReqs: Maximale Aufladerate, Batteriekapazität
 - SoftReqs: Minimale quadratische Differenz der aufeinanderfolgenden Records.
- Schlussfolgerung
 - Vorhersage vs Aktionen. Viele verschiedene Requirements. Beeindruckende Experimentergebnisse.

Theory Guided Data Science

- Scientific Data vs Internet Data
 - Scientific Data: Experiments, Observations, Simulations
 - Object is understanding causalities (eg why is drug effective). Many variables with many unclear dependencies. #data often does not suffice for automated model building.
 - Internet Data: User Behaviour, Webcontent
 - ~2x larger than scientific data. Objective is typically operating a service (eg ad placement)
- Theory Guided Data Science: Domain experts already have models (eg equations, strategy spaces, ...) with limitations (not full coverage, simplification, unclear whether model is correct)



- Pure data science approach is not enough. Tight-couple data-analysis models with conventional scientific models
- Topic: What can this mean? Which objectives have been identified? What has been tried so far? Classification of existing approaches.
- Understanding causalities => better predictions
- Physically constituent models: Predictionmodel in line with scientific laws. Performance + Accuracy + Simplicity + Consistency
- Classifications of Approaches: Choice of model structure, using domain knowledge to tune models, refine model output with domain knowledge, hybrid models, data based modelrefinement. Core teaching objective is this classification.
 - Choice of model structure
 - "Theory guided design of data-science models". Which model for given problem?
 - Domain knowledge suggests structure.
 - Linear Regression (but objects usually are not linear), Generalized Linear Model (GLM, more flexible for dependencies between input and response)
 - "Theory guided specification of Response".
 - 1) Appropriate choice of error distribution.
 - 2) Instantiate GLM appropriately
 - Choice of Model Architecture
 - Modularize problem adequately regarding domain knowledge, construct model for each subproblem separately.
 - Learning of Model Architecture that is adequate
 - Using domain knowledge to tune the models
 - Initialization of approaches
 - Example: Result of k-Means depends on init.
 - Previous approach: Seeds must be data objective. Alternative (mind game): Choose partitions of seeds randomly, but only physically consistent points.
 - Matrix completion
 - ◆ Incomplete data matrix (eg Person x Filmbewertung), fill empty spots (predict Filmbewertung per Person)
 - ◆ Many approaches, e.g. find NN, other approach: Iterative, per cell: init empty cells with random values, get better values. global averages as init alternatively (good init => quick calculations)
 - ◆ Domain knowledge from biology: Tree-of-Life. Dann können statt globalem Durchschnitt konkret ausgewählte Zeilen verwendet werden
 - Restricting possible models with domain knowledge
 - Bayes Network: Graph representation of variables and relationships between (eg guter Kunde/schlechter Kunde vorhersagen)
 - Objective: Adequate consideration of dependencies of input variables for prediction (Correlations)
 - Not practical to take in all correlations, approach for constructing bayes network: specify attribute pairs whose correlation should be taken into account.
 - Constraint-Based Pattern Mining
 - ◆ Previous explanation: Constraints represent interest. Now they represent background knowledge. Eg: Constraint "Symptome A, B, are specific to old age".
 - Complex Constraints: Differentialgleich z.B. Wie berücksichtigen? Forschungsfrage.
 - Constraints to ensure physical consistency
 - Example: Spot surface bodies of water. Poor data quality (noise, missing values, ...). Elevation profile of area is helpful.
 - Domain knowledge to control regularization
 - Ziel: Simpler models with background knowledge.
 - Already Addressed: Leaving domain knowledge aside, pruning decision trees to cope with overfitting.
 - Lasso: Linear model, numeric prediction. Prediction is linear combination of vars. Simplicity is important
 - ◆ Previous quality measures (eg mean absolute error) only cover predictive power, not simplicity)
 - ◆ New quality measure "lasso": PredictivePowerQuality + #variables needed
 - ◆ Choice of vars arbitrary, only number counts
 - Group Lasso: Arbitrary variables not ideal. Partition vars into groups, only entire group as model variables
 - ◆ Partition requires domain knowledge. Which variables in combination might serve explanation?
 - ◆ Groups may overlap - "Graph lasso": Variables are nodes, edges when variables in combination may form explanation. If path between vars, they may be part of the model.
 - ◆ Space is being structured apriori by hand in domain specific manner

- Refine model output using domain knowledge
 - Knowledge explicitly known
 - Example: Find materials with specific characteristics. Systematic search practically not doable because too many parameters.
 - Alternative approach: Prediction of characteristics of materials with appropriate data-analysis methods
 - ◆ Relies on trainingdata ie materials with known features. Ab initio calculations (characteristics computation based on structural characteristics using certain model) only for promising candidates.
 - Knowledge implicitly known (muss zuerst hergeleitet werden)
 - Example: Mehrere Fotos von derselben Wasser/Erdfläche zu verschiedenen Zeiten verwenden um Höhenprofil zu lernen.
 - Two directions:
 - ◆ Better classification of image content with help of elevation profile
 - ◆ Better elevaton profile by analyzing images
 - Iterative approach: Each step has best results so far as input (immer zwischen beiden Richtungen wechseln und den einen Output als nächsten Input nehmen)
- Hybrid Models
 - Partly conventional scientific models, partly data-oriented
 - Build large model from smaller models. Find problems with existing models.
 - Scientists already have models, they are a) simplifications, b) cover only aspects of a process.
 - Approach a) (Simplification): Suche nach Auffälligkeiten und Schwächen des aktuellen Models: Delta von predictions by scientific model und tatsächliches Systemverhalten
 - Approach b): Modularize problem, construct models for subproblems. See "Choice of Model Architecture"
- Data-based refinement/calibration of models
 - GLUE: Choice of good parameter values for a given model (Calibration)
 - Example: Hydrologists. One wants to predict max water runoff per time. Measuredata from past is available, differentiation between calibration events and validation data.

Storm no.	Date	Total Rainfall (mm)	Maximum intensity (mm h ⁻¹)	Peak flow
1	17-19 Nov 81	80.51	9.17	8.0
2	27-29 Jan 83	111.44	8.60	6.1
3	11-13 Feb 76	107.25	10.43	8.5
4	5-7 Aug 73	121.77	25.66	16.8*
5	17-19 Nov 78	124.18	8.74	7.6
6	8-10 Dec 83	98.23	8.72	7.2
7	5-7 Oct 80	94.78	13.34	10.5
8	13-15 Jan 81	74.91	11.56	8.3
9	19-22 Apr 78	75.60	5.86	3.70
10	20-22 Jul 81	55.30	5.54	2.21

* Estimated

- Models have parameters such as "hydraulic conductivity" of soil. Impossible to find.

- Model Components

- Qualitätsmaß, um zu quantifizieren, wie gut Vorhersage einer Größe ist (zB Delta Vorhersage - Messung)
- Qualitätsmaß, um gesamte Vorhersage über alle Größen ist (zB gewichteter Durchschnitt über alle Größenvorhersagen)
- Wertebereich der Parameter? Wahrscheinlichkeitsverteilung der Parameter? Gut gewählte Initialerteilung beschleunigt Berechnungen.
- Given specific parameter values&distributions, one can compute: Prediction, prediction likelihood.
- Jetzt bekannt: Gegeben einer Beobachtung, wie wahrscheinlich ist Kombination von Parameterwerten?
 - ◆ Iterative Approach: Resampling of param values, based on given distributions. Compute probabilities of predictions, based on comparison with actual observations, recompute probability distributions of param values.

- Schlussbemerkungen:

- Motivation: Conventional data-analysis methods are not always productive in science: Not enough examples, *results not new from user experience*.
- Abgrenzung vom "Internet data" Scenario
- Existing scientific models tend to be imperfect.
- Objective of TGDS: Better models by tight coupling of existing models/domain knowledge and data-analysis models
- Physically consistent models.
- Viele Alternativen.
- Topic of this chapter: Overview of currently existing approaches and their classification.
- How useful are various alternatives? Calls for evaluation regarding effort and utility.
- How should domain expert proceed when doing TGDS?
- Outlook/vision: Classification -> Method -> System support.