Wissensentdeckung in Datenbanken

SQL, Häufige Mengen

Nico Piatkowski und Uwe Ligges

Informatik—Künstliche Intelligenz
Computergestützte Statistik
Technische Universität Dortmund

11.05.2017

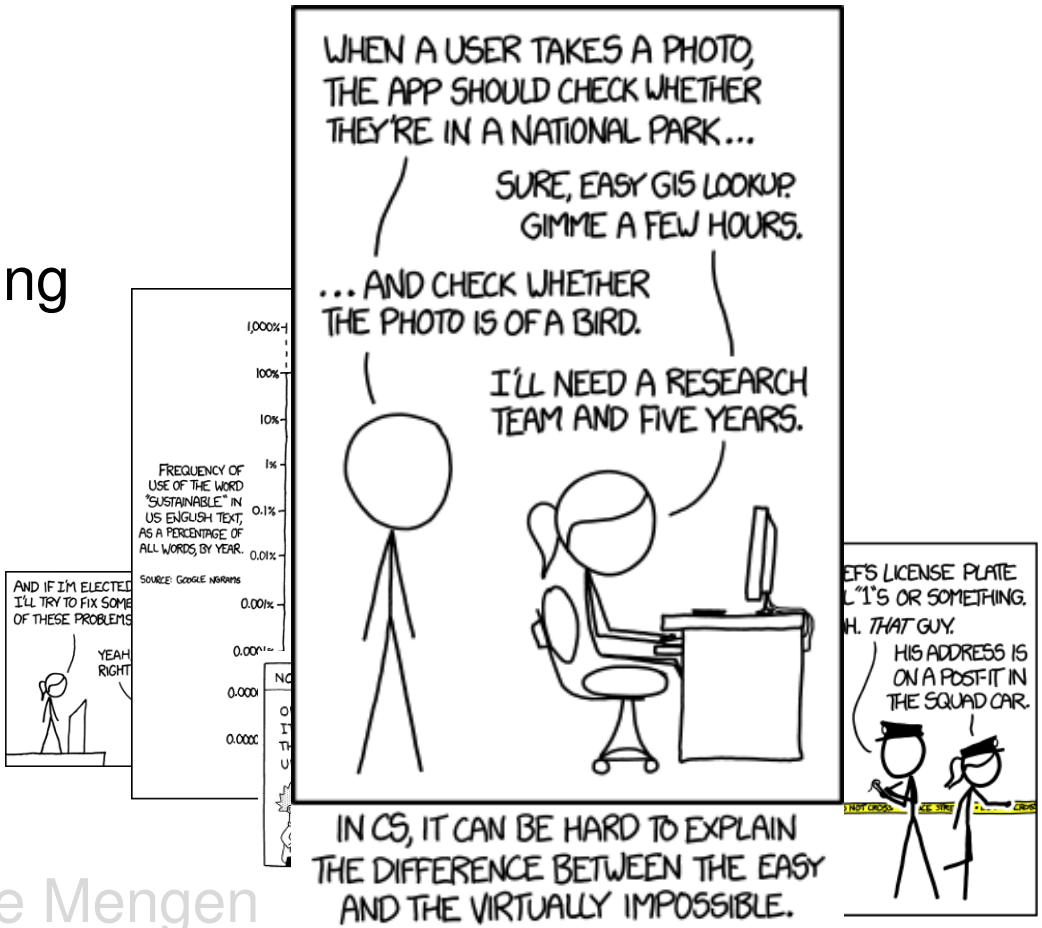
Überblick

Was bisher geschah...

- Modellklassen
- Verlustfunktionen
- Numerische Optimierung
- Regularisierung
- Überanpassung
- SQL

Heute

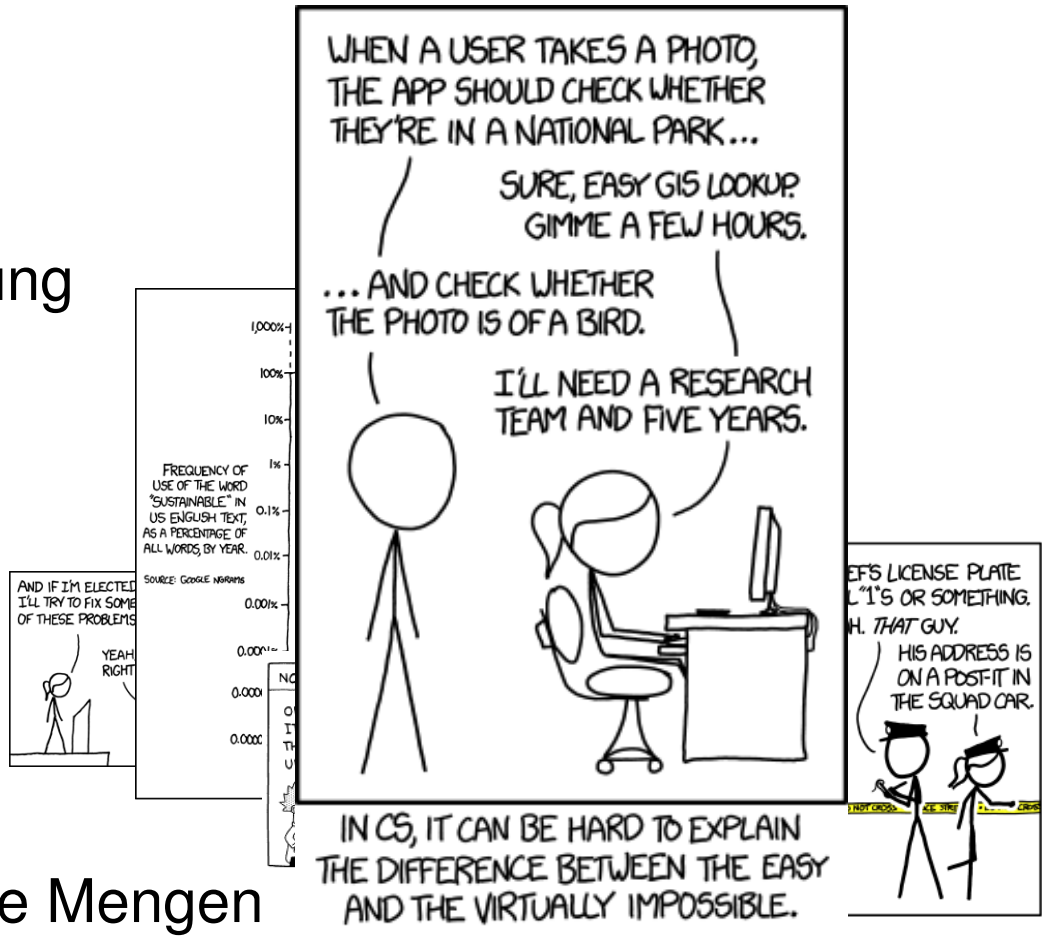
- SQL (Forts.)
- Algorithmen für häufige Mengen
- Komprimierende Muster



Überblick

Was bisher geschah...

- Modellklassen
- Verlustfunktionen
- Numerische Optimierung
- Regularisierung
- Überanpassung
- SQL



Heute

- SQL (Forts.)
- Algorithmen für häufige Mengen
- Komprimierende Muster

Datenbanken und SQL

Jetzt: $\mathcal{M} \subset \mathcal{D}$ bzw. $\mathcal{M} \subset \mathcal{X}$

- Relationale Datenbanken ≡ Menge von Tabellen
- Relationales Datenbankmanagementsystem erlaubt Anfrage und Manipulation von Daten mittels Structured Query Language (SQL)
- Beispielhafte Anfragen `Q`:
  - `SELECT DISTINCT x1,x2 FROM data WHERE ...`
  - `SELECT * FROM ... ORDER BY x`
  - `SELECT AVG(x) FROM ... GROUP BY y`
  - `SELECT MIN(x) FROM ... GROUP BY y`
  - `SELECT MAX(x) FROM ... GROUP BY y`
  - `SELECT COUNT(x) AS c FROM ... HAVING c>10`

Datenbanken und SQL

Jetzt: $\mathcal{M} \subset \mathcal{D}$ bzw. $\mathcal{M} \subset \mathcal{X}$

- Relationale Datenbanken ≡ Menge von Tabellen
- Relationales Datenbankmanagementsystem erlaubt Anfrage und Manipulation von Daten mittels Structured Query Language (SQL)
- Beispielhafte Anfragen `Q`:
  - `SELECT DISTINCT x1,x2 FROM data WHERE ...`
  - `SELECT * FROM ... ORDER BY x`
  - `SELECT AVG(x) FROM ... GROUP BY y`
  - `SELECT MIN(x) FROM ... GROUP BY y`
  - `SELECT MAX(x) FROM ... GROUP BY y`
  - `SELECT COUNT(x) AS c FROM ... HAVING c>10`

Beispiel: SQLite

Name	Monat	Jahr	Stadt
Tom	3	1981	München
Christian	2	1993	Freiburg
Lukas	5	1996	Köln
Stefan	8	1991	Mainz
Nicolas	4	1990	Freiburg
Leonardo	12	1993	Köln
Zlatko	9	1987	Bremen
Marcel	3	1994	Leipzig
Anthony	4	1988	Köln
Florian	10	1990	Freiburg
Fin	2	1987	Bremen

```
CREATE TABLE elf(
Name TEXT, Monat INT,
Jahr INT, Stadt TEXT);

.mode csv
.import elf.csv elf
```

Beispiel: SQLite

Name	Monat	Jahr	Stadt
Tom	3	1981	München
Christian	2	1993	Freiburg
Lukas	5	1996	Köln
Stefan	8	1991	Mainz
Nicolas	4	1990	Freiburg
Leonardo	12	1993	Köln
Zlatko	9	1987	Bremen
Marcel	3	1994	Leipzig
Anthony	4	1988	Köln
Florian	10	1990	Freiburg
Fin	2	1987	Bremen

```
SELECT * FROM elf
WHERE 2017-Jahr >= 30
AND Monat <= 5;
```

Beispiel: SQLite

Name	Monat	Jahr	Stadt
Tom	3	1981	München
Christian	2	1993	Freiburg
Lukas	5	1996	Köln
Stefan	8	1991	Mainz
Nicolas	4	1990	Freiburg
Leonardo	12	1993	Köln
Zlatko	9	1987	Bremen
Marcel	3	1994	Leipzig
Anthony	4	1988	Köln
Florian	10	1990	Freiburg
Fin	2	1987	Bremen

```
SELECT * FROM elf
WHERE Stadt = "Köln";
```

Beispiel: SQLite

Name	Monat	Jahr	Stadt
Tom	3	1981	München
Christian	2	1993	Freiburg
Lukas	5	1996	Köln
Stefan	8	1991	Mainz
Nicolas	4	1990	Freiburg
Leonardo	12	1993	Köln
Zlatko	9	1987	Bremen
Marcel	3	1994	Leipzig
Anthony	4	1988	Köln
Florian	10	1990	Freiburg
Fin	2	1987	Bremen

```
SELECT * FROM elf
WHERE
Stadt LIKE "%ei%";
```

Beispiel: SQLite

Name	Monat	Jahr	Stadt
Tom	3	1981	München
Christian	2	1993	Freiburg
Lukas	5	1996	Köln
Stefan	8	1991	Mainz
Nicolas	4	1990	Freiburg
Leonardo	12	1993	Köln
Zlatko	9	1987	Bremen
Marcel	3	1994	Leipzig
Anthony	4	1988	Köln
Florian	10	1990	Freiburg
Fin	2	1987	Bremen

```
SELECT * FROM elf
WHERE
Stadt LIKE "%g";
```

Beispiel: SQLite

Name	Monat	Jahr	Stadt
Tom	3	1981	München
Christian	2	1993	Freiburg
Lukas	5	1996	Köln
Stefan	8	1991	Mainz
Nicolas	4	1990	Freiburg
Leonardo	12	1993	Köln
Zlatko	9	1987	Bremen
Marcel	3	1994	Leipzig
Anthony	4	1988	Köln
Florian	10	1990	Freiburg
Fin	2	1987	Bremen

```
SELECT * FROM elf
WHERE
Stadt NOT LIKE "%n%";
```

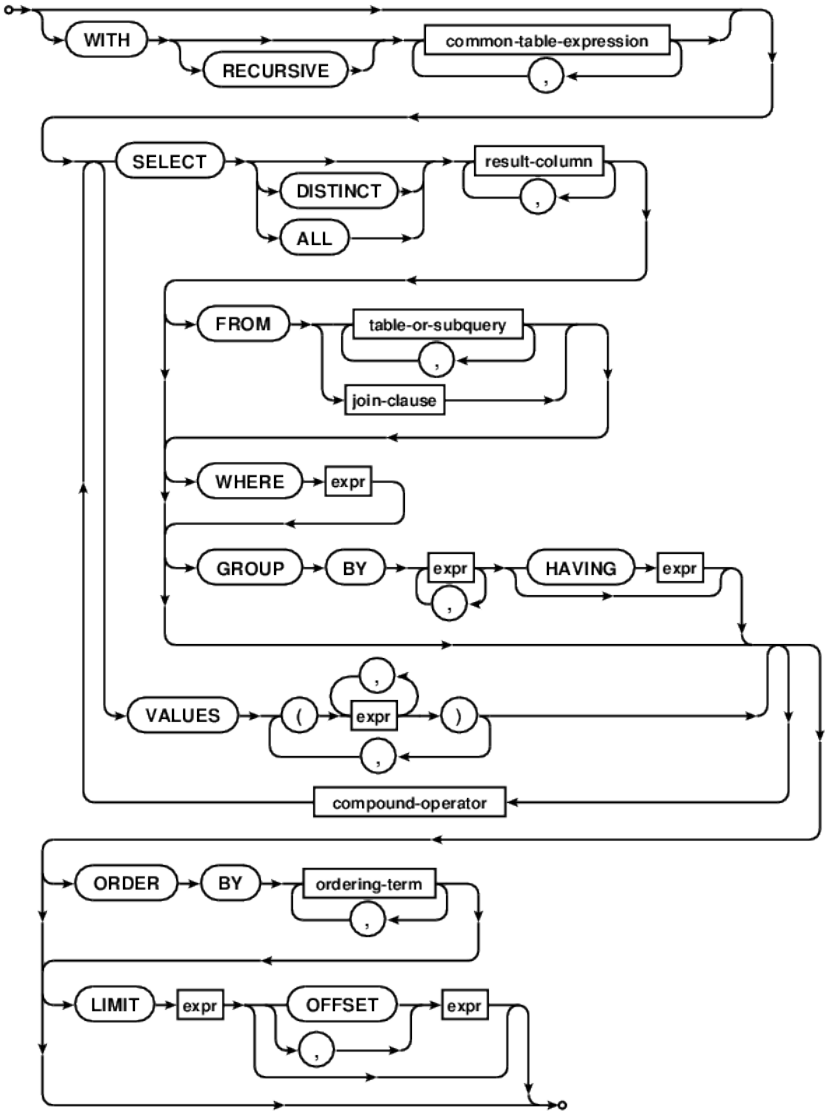
Beispiel: SQLite

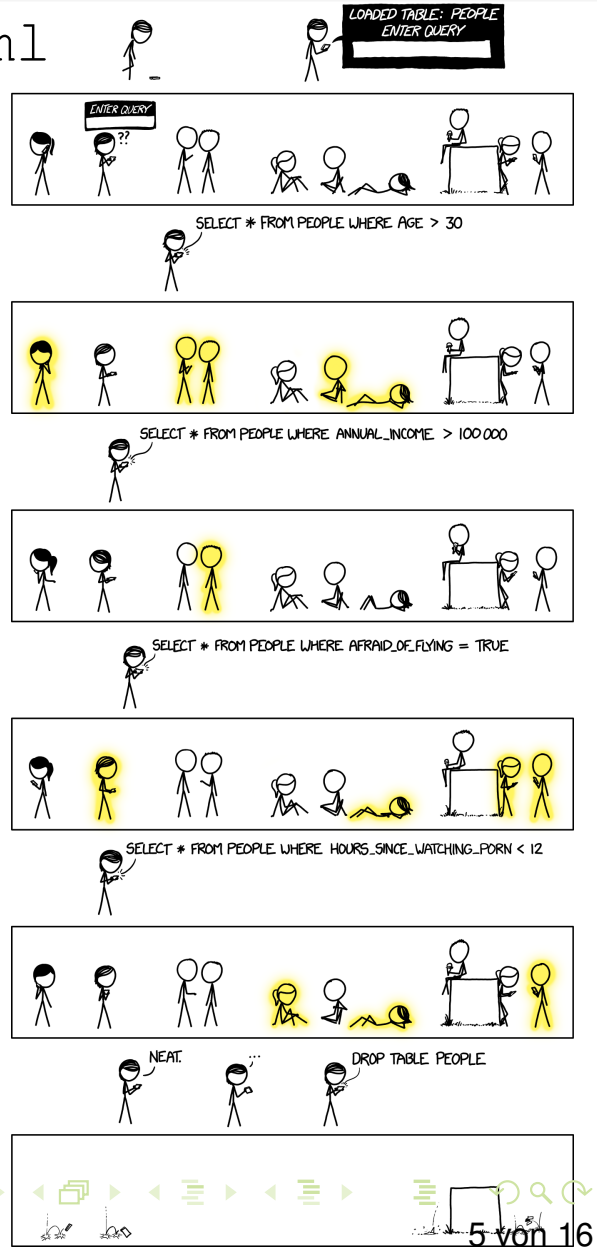
Name	Monat	Jahr	Stadt
Tom	3	1981	München
Christian	2	1993	Freiburg
Lukas	5	1996	Köln
Stefan	8	1991	Mainz
Nicolas	4	1990	Freiburg
Leonardo	12	1993	Köln
Zlatko	9	1987	Bremen
Marcel	3	1994	Leipzig
Anthony	4	1988	Köln
Florian	10	1990	Freiburg
Fin	2	1987	Bremen

```
SELECT * FROM elf
WHERE
UPPER(Name) LIKE "%C%"
OR Jahr = 1990;
```

SQLite SELECT Anweisung

https://www.sqlite.org/lang_select.html





Anfragen und Mengen

Resultat einer Datenbankanfrage $\mathtt{Q}$ ist eine neue Tabelle $\mathtt{D}$

- Annahme: Datenbank besteht aus Binärdaten, oder wird mittels SQL Anfragen konvertiert
- Die Einträge von $\mathtt{D}$ heißen dann *Transaktionen*
- Transaktion $t \in \mathtt{D}$ entspricht Indikatorvektor einer Menge

$$00101001110100011001 0 \equiv \{x_3, x_5, x_8, x_9, x_{10}, x_{12}, x_{16}, x_{17}, x_{20}\}$$

- Die Elemente x_i der Menge nennt man auch *Items*

Anfragen und Mengen

Resultat einer Datenbankanfrage $\mathtt{Q}$ ist eine neue Tabelle $\mathtt{D}$

- Annahme: Datenbank besteht aus Binärdaten, oder wird mittels SQL Anfragen konvertiert
- Die Einträge von $\mathtt{D}$ heißen dann *Transaktionen*
- Transaktion $t \in \mathtt{D}$ entspricht Indikatorvektor einer Menge

$$00101001110100011001 0 \equiv \{x_3, x_5, x_8, x_9, x_{10}, x_{12}, x_{16}, x_{17}, x_{20}\}$$

- Die Elemente x_i der Menge nennt man auch *Items*

Häufige Mengen

- Der *Support* einer (Teil-)Menge X entspricht der Häufigkeit von X in $\mathtt{D}$

$$\mathrm{supp}(X) = |\{t \in \mathtt{D} \mid X \subseteq t\}|$$

- **Frequent Itemset Mining**: Wähle $\epsilon > 0$ und bestimme alle Mengen $X \subseteq \{x_1, x_2, \ldots, x_n\}$ mit $\mathrm{supp}(X) \geq \epsilon$
  - Solche Menge heißen *häufig*
- Insgesamt 2^n mögliche häufige Mengen
- $\Rightarrow$ Hoher (inhärenter) Ressourcenbedarf für das Berechnen und Speichern aller Häufigkeiten
- Worst-case Komplexität unabhängig vom Algorithmus
- Aber: Große Mengen sind selten häufig

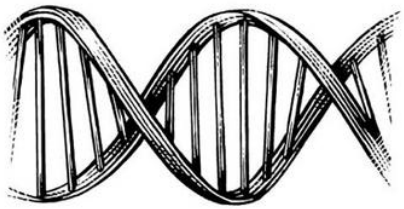
Häufige Mengen

- Der *Support* einer (Teil-)Menge X entspricht der Häufigkeit von X in $\mathtt{D}$

$$\mathrm{supp}(X) = |\{t \in \mathtt{D} \mid X \subseteq t\}|$$

- **Frequent Itemset Mining**: Wähle $\epsilon > 0$ und bestimme alle Mengen $X \subseteq \{x_1, x_2, \ldots, x_n\}$ mit $\mathrm{supp}(X) \geq \epsilon$
  - Solche Menge heißen *häufig*
- Insgesamt 2^n mögliche häufige Mengen
- $\Rightarrow$ Hoher (inhärenter) Ressourcenbedarf für das Berechnen und Speichern aller Häufigkeiten
- Worst-case Komplexität unabhängig vom Algorithmus
- Aber: Große Mengen sind selten häufig

Anwendungen



- Häufige Mutationen in DNS
- Zusammenfassen von Textkorpora
- Warenkorbanalyse
- Nutzungsdaten/Logdaten

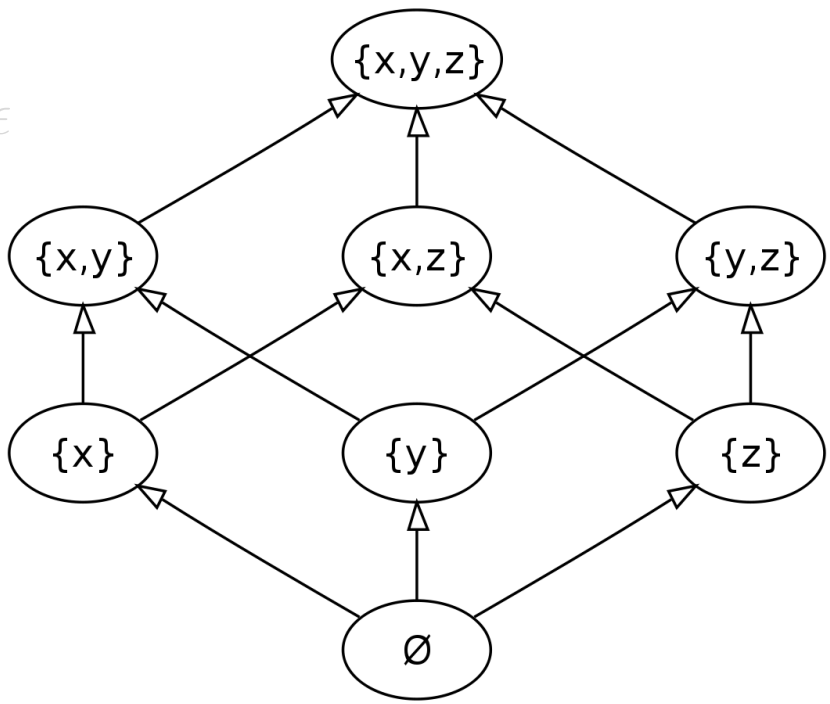
Apriori

- Anti-Monotonie Eigenschaft: $\forall X \subseteq \{x_1, x_2, \ldots, x_n\}$

$$X \subseteq Y \Rightarrow \text{supp}(Y) \leq \text{supp}(X)$$

Apriori Algorithmus (Eingabe: $\mathtt{D}, \epsilon$)

1. $i \leftarrow 1; x_i \in W_1 \Leftrightarrow \text{supp}(\{x_i\}) \geq \epsilon$
2. Füge alle $X \subseteq \bigcup_{U \in W_i} U$ mit $|X| = i + 1$ und $\text{supp}(X) \geq \epsilon$ in W_{i+1} ein
3. $i \leftarrow i + 1$
4. Falls $i < n \wedge W_i \neq \varnothing$: Goto 2
5. Ausgabe: $\bigcup_{j=1}^{i} W_j$



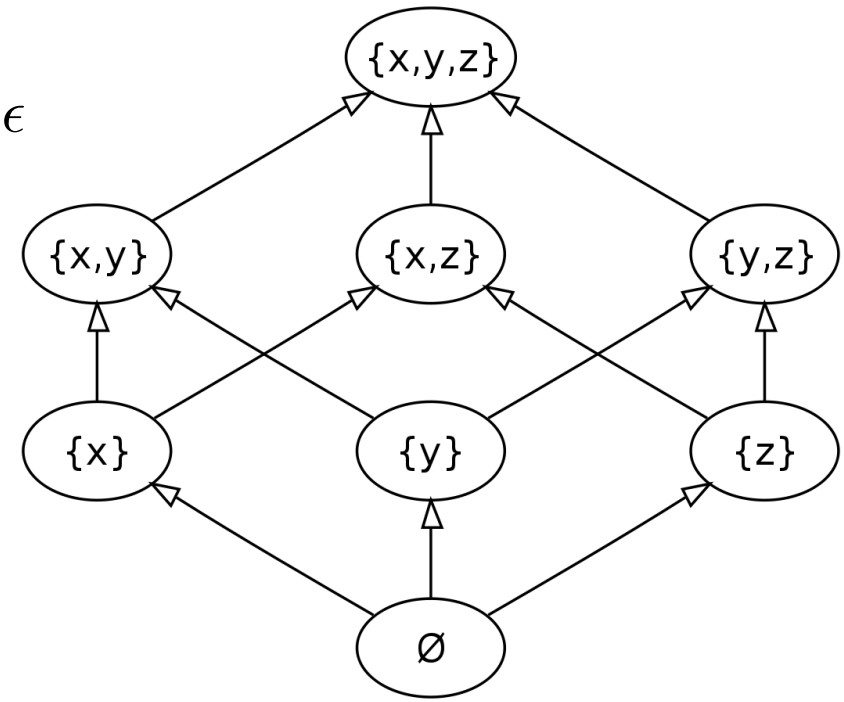
Apriori

- Anti-Monotonie Eigenschaft: $\forall X \subseteq \{x_1, x_2, \ldots, x_n\}$

$$X \subseteq Y \Rightarrow \text{supp}(Y) \leq \text{supp}(X)$$

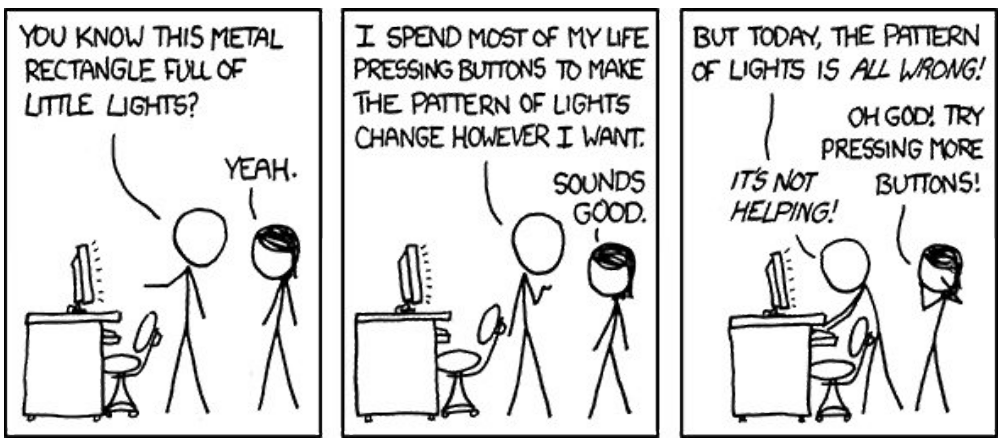
Apriori Algorithmus (Eingabe: $\mathtt{D}, \epsilon$)

1. $i \leftarrow 1$; $x_i \in W_1 \Leftrightarrow \text{supp}(\{x_i\}) \geq \epsilon$
2. Füge alle $X \subseteq \bigcup_{U \in W_i} U$ mit $|X| = i + 1$ und $\text{supp}(X) \geq \epsilon$ in W_{i+1} ein
3. $i \leftarrow i + 1$
4. Falls $i < n \wedge W_i \neq \varnothing$: Goto 2
5. Ausgabe: $\bigcup_{j=1}^{i} W_j$



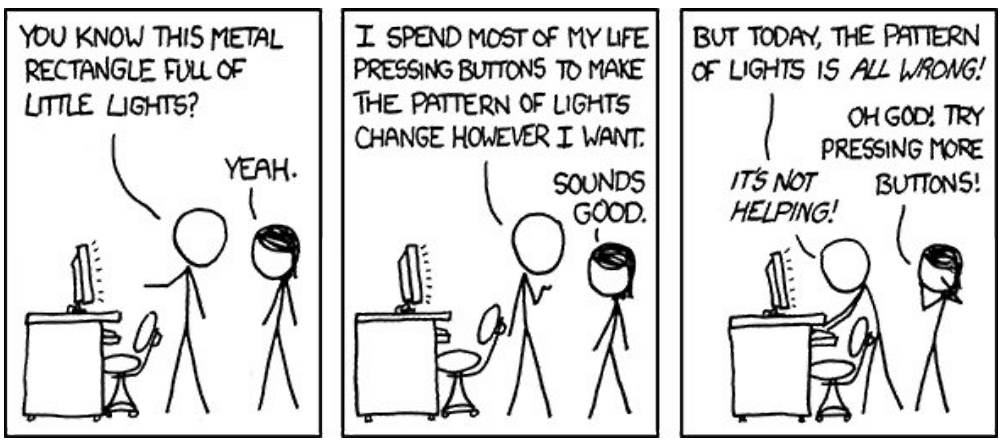
Apriori (II)

- Einfach zu implementierender Bottom-Up Algorithmus
- Apriori kann die Anzahl der zu betrachtenden Kandidaten stark reduzieren
- Falls ϵ “klein” (≤ 1), viele Kandidaten möglich
- Falls $|X|$ “groß” ($\approx n$), viele Kandidaten möglich
- n Scans der kompletten Datenbank erforderlich
- Komplexität Implemetierungsunabhängig



Apriori (II)

- Einfach zu implementierender Bottom-Up Algorithmus
- Apriori kann die Anzahl der zu betrachtenden Kandidaten stark reduzieren
- Falls ϵ “klein” (≤ 1), viele Kandidaten möglich
- Falls $|X|$ “groß” ($\approx n$), viele Kandidaten möglich
- n Scans der kompletten Datenbank erforderlich
- Komplexität Implemetierungsunabhängig



FP-growth

- Spezialisierte Datenstruktur (Frequent Pattern Tree)
  - Basiert auf Prefixbaum
  - Knoten entsprechen häufige Mengen der Größe 1
  - Pfade entsprechen größeren häufige Mengen
  - Baumstruktur repräsentiert Häufigkeiten
- Rekursiver Algorithmus zur Berechnung der Häufigkeit beliebiger Mengen
- Idee:
  - Reduziere jede Transaktion t auf die Menge der häufigen Items
  - Nutzung eine Ordnung auf den Items um jeder Transaktionen einen eindeutigen String zuzuordnen
  - Häufigkeiten identischer Teilstrings können dann in einem Prefixbaum zusammengefasst werden

FP-growth

- Spezialisierte Datenstruktur (Frequent Pattern Tree)
  - Basiert auf Prefixbaum
  - Knoten entsprechen häufige Mengen der Größe 1
  - Pfade entsprechen größeren häufige Mengen
  - Baumstruktur repräsentiert Häufigkeiten
- Rekursiver Algorithmus zur Berechnung der Häufigkeit beliebiger Mengen
- Idee:
  - Reduziere jede Transaktion t auf die Menge der häufigen Items
  - Nutzung eine Ordnung auf den Items um jeder Transaktionen einen eindeutigen String zuzuordnen
  - Häufigkeiten identischer Teilstrings können dann in einem Prefixbaum zusammengefasst werden

FP-growth

- Spezialisierte Datenstruktur (Frequent Pattern Tree)
  - Basiert auf Prefixbaum
  - Knoten entsprechen häufige Mengen der Größe 1
  - Pfade entsprechen größeren häufige Mengen
  - Baumstruktur repräsentiert Häufigkeiten
- Rekursiver Algorithmus zur Berechnung der Häufigkeit beliebiger Mengen
- Idee:
  - Reduziere jede Transaktion t auf die Menge der häufigen Items
  - Nutzung eine Ordnung auf den Items um jeder Transaktionen einen eindeutigen String zuzuordnen
  - Häufigkeiten identischer Teilstrings können dann in einem Prefixbaum zusammengefasst werden

FP-tree

- Die Wurzel ist “leer”
- Alle anderen Knoten sind 3-Tupel: $(\text{name}, \text{count}, \text{link})$
- Jeder Pfad entspricht einer häufigen Menge
- Eine Sprungtabelle erlaubt den schnellen Zugriff auf alle Itemsets die ein bestimmtes Item enthalten

TID	Menge	String ($\epsilon = 3$)
1	$\{f, a, c, d, g, i, m, p\}$	fcamp
2	$\{a, b, c, f, l, m, o\}$	fcabm
3	$\{b, f, h, j, o\}$	fb
4	$\{b, c, k, s, p\}$	cbp
5	$\{a, f, c, e, l, p, m, n\}$	fcamp

FP-tree

- Die Wurzel ist “leer”
- Alle anderen Knoten sind 3-Tupel: $(\text{name}, \text{count}, \text{link})$
- Jeder Pfad entspricht einer häufigen Menge
- Eine Sprungtabelle erlaubt den schnellen Zugriff auf alle Itemsets die ein bestimmtes Item enthalten

TID	Menge	String ($\epsilon = 3$)
1	$\{f, a, c, d, g, i, m, p\}$	fcamp
2	$\{a, b, c, f, l, m, o\}$	fcabm
3	$\{b, f, h, j, o\}$	fb
4	$\{b, c, k, s, p\}$	cbp
5	$\{a, f, c, e, l, p, m, n\}$	fcamp

FP-tree (II)

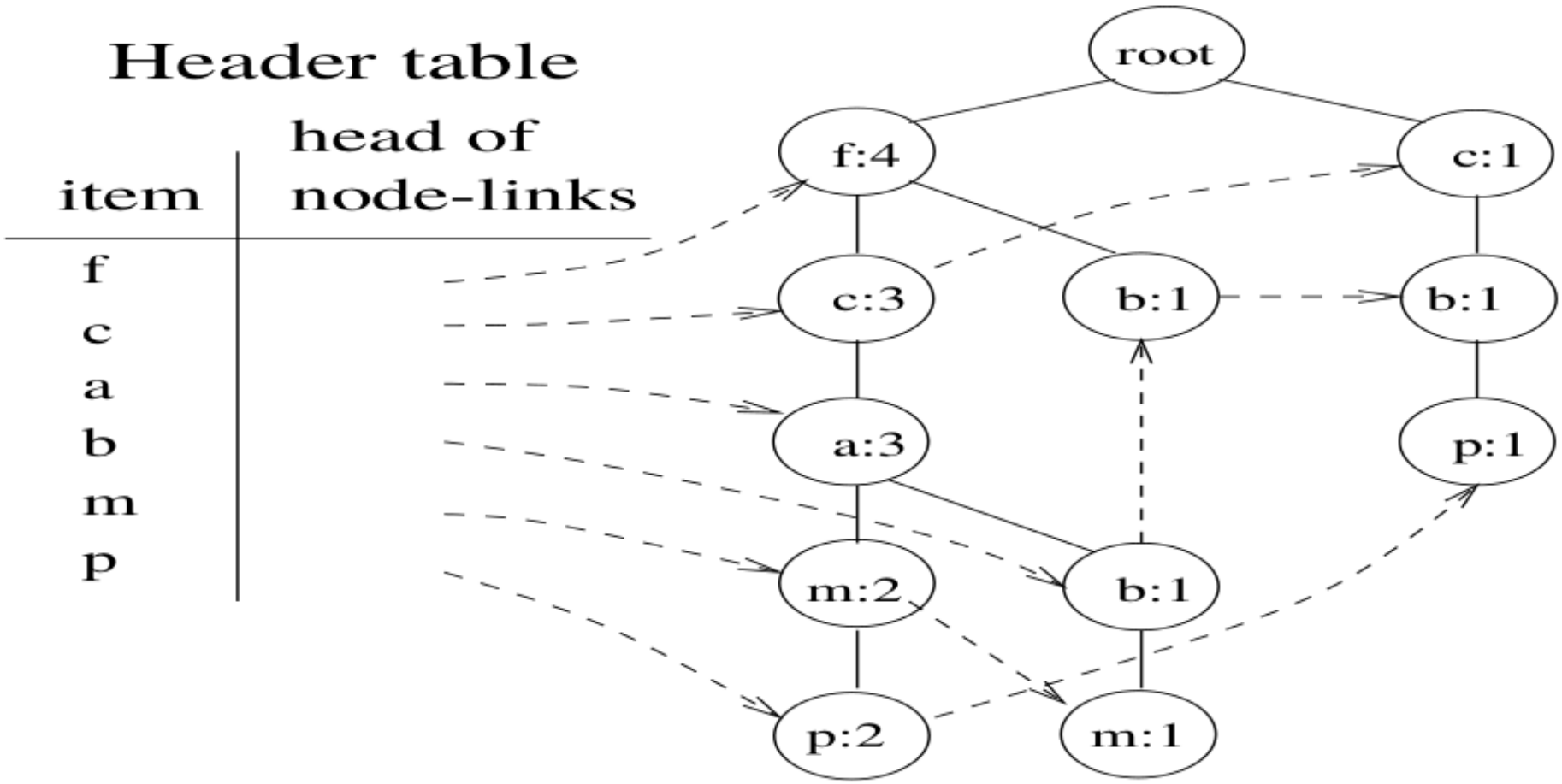


Figure 1: The FP-tree in Example 1.

FP-growth (II)

- Einfach zu implementierender Bottom-Up Algorithmus
- Laufzeit $2|\mathrm{D}||t_{\max}|$
- Tiefe des Baums is $|t_{\max}| + 1$
- FP-tree höchstens so groß wie Datenbank (ein Pfad pro Transaktion)
  - Im Gegensatz zu Apriori!
- 2 Scans der kompletten Datenbank erforderlich
- Häufigkeiten **aller** Itemsets können dann rekursiv aus dem FP-tree abgelesen werden

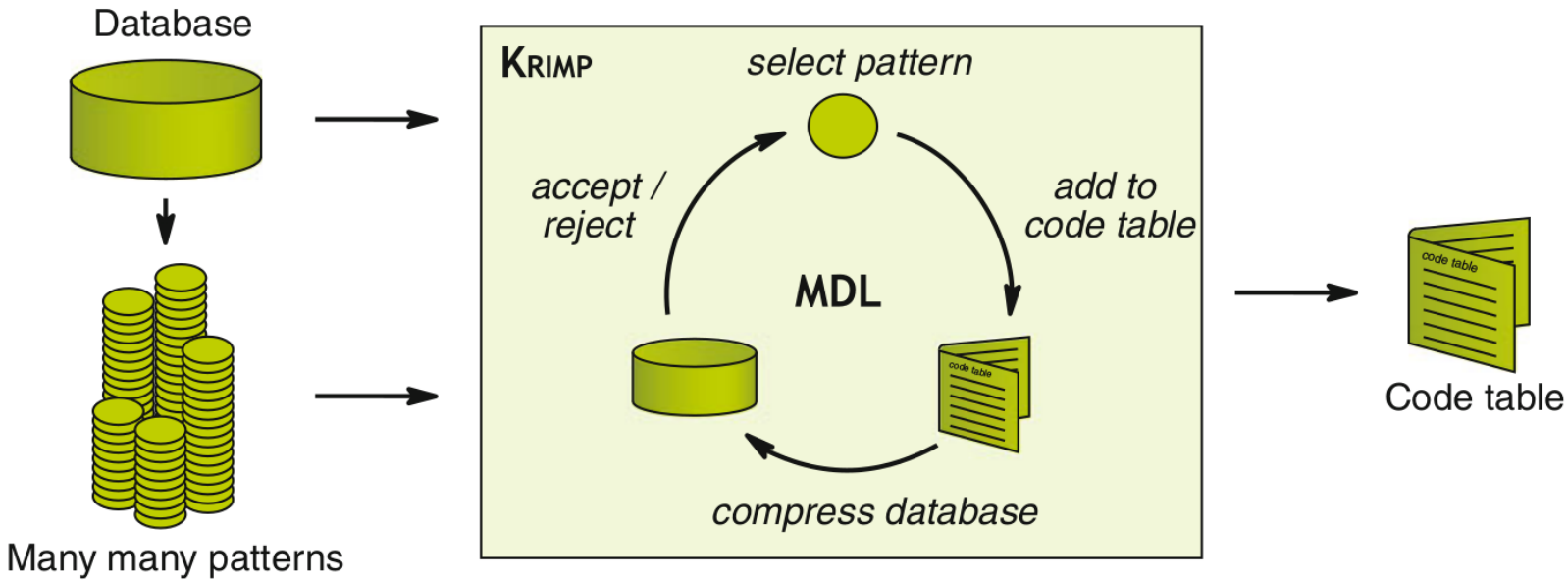
Qualität häufiger Mengen

- “Pattern Explosion”: Große Datenbanken beinhalten exponentiell viele häufige Mengen
- Viele Mengen sind redundant
- Viele Mengen sind nicht “interessant”
- Oft: Zusätzliche Nachbearbeitung bzw. Filtern der Ergebnisse, um nicht relevante Mengen aus dem Ergebnis zu entfernen
- Aber: Kein generisches Maß für “Interessantheit” verfügbar

MDL und KRIMP

- Minimum Description Length (MDL)
- Formalisierung von Ockhams Rasiermesser

$$\min_{C \in \mathcal{M}} L(C) + L(\mathcal{D} \mid C)$$



Literatur: Jilles Vreeken, Matthijs van Leeuwen und Arno Siebes. KRIMP: mining itemsets that compress. 2011