Begriffe

Artikel	$a,b,c,.. \in U$	
Universum	U	$\{a,b,c,d,e,...\}$
Transaktion	$T \in \wp(U)$	$\{b,c,e,f,i\}$
Artikelmengen	$X, Y, ... \subseteq T$	$\{c,f,i\}$
Datenbasis	$D \subseteq_{mul} \wp(U)$	$\{\{a,b,c\}\{a,b\}\{a,b\}\{b\}...\}$
Assoziationsregel	$R: X \Rightarrow Y$	$ab \Rightarrow de$

Annahme: T, X, Y sind geordnet

Assoziationsregeln – wozu??

Kunden,
die **A**ftershave, **B**ier, und **C**hips *kaufen, kaufen oft auch*
Dosensuppen und **E**ier

$$abc \Rightarrow de$$

⇒ „Günstige“ Plazierung von Waren im Kaufhaus

Einschränkung:
Artikelmengen sind binär,
d.h. Artikel sind „eingepackt“ oder „nicht eingepackt“

Größen

Confidence: „Vertrauen“ in die Regel $X \Rightarrow Y$
= Anzahl aller T mit: $X \subseteq T \Rightarrow Y \subseteq T$

$$confidence(R) = \frac{\left|\{T \mid (X \cup Y) \subseteq T\}\right|}{\left|\{T \mid X \subseteq T\}\right|},\ T \subseteq D$$

Support: „Unterstützung“ der Regel $X \Rightarrow Y$
= Anzahl aller T mit $(X \cup Y) \subseteq T$

$$support(R) = \frac{\left|\{T \mid (X \cup Y) \subseteq T\}\right|}{|D|},\ T \subseteq D$$

Vorgehensweise

⇒ Minimaler Support *minsup* **und** Minimale Confidence *minconf*
⇒ Zerlegung in zwei Teilprobleme:

1. Finde alle Regeln R_i mit $support(R_i) \geq minsup$
2. Finde in dieser Regelmenge alle R_j
   mit $confidence(R_i) \geq mincon$

Der Apriori-Algorithmus

Definition: **k-Artikelmenge** = Artikelmenge der Länge k
„häufige" Artikelmenge: X mit $support(X) \geq minsup$
L_k = Menge aller häufigen k-Artikelmengen

Aufgabe: Finde alle „häufigen" Artikelmengen
Idee: Wenn X häufig, so auch $Y \subseteq X$
Benutze L_{k-1} als „Saat" für L_k
Algorithmus:
1. Generiere alle großen 1-Artikelmengen = L_1
2.Generiere (*a priori*) Kandidatenmenge C_k aus L_{k-1}
3. L_k ={Alle $X_i \in C_k$ mit ausreichend Support in D}
4. Gehe zu 2.

apriori-gen

Naiver Ansatz: Hinzufügen aller Artikel $i \in U$ zu allen $X_i \in L_{k-1}$
$\Rightarrow$ erzeugt zu viele Kandidaten

Idee: Wenn $X \in L_k$ dann sind alle $Y_i \subseteq X$ in L_{k-1}
Algorithmus: 1. Bilde Vereinigungsmengen:
$Z_i = X_j \cup Y_k$
$\forall X_j, Y_k \in L_{k-1}$ deren erste k-2 Stellen gleich sind
2. Verwerfe Z_i mit: $\exists V \subseteq Z_i \notin L_{k-1}$
3. $C_k = \{Z_i\}$

Beispiel apriori-gen

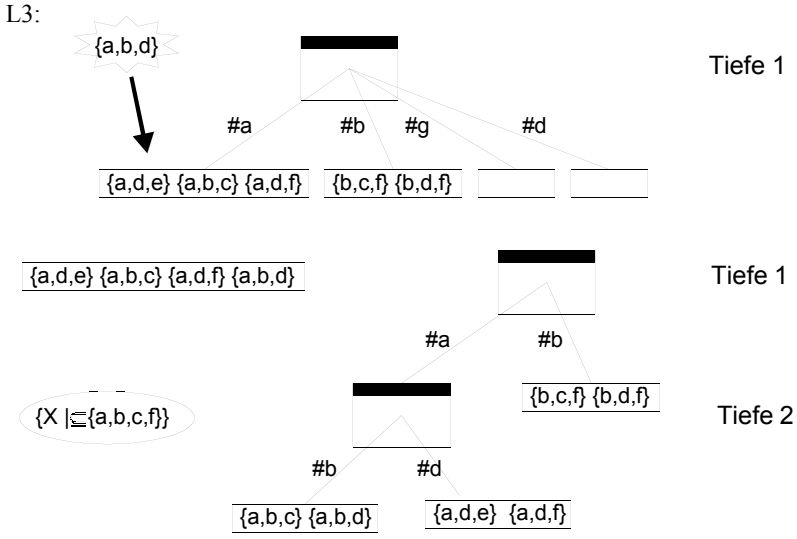
Warum müssen die ersten k-2 Stellen stimmen?

k=5

L_4={{abcd, {abce}, {abde}, {acde}, {bcde}}
$\Rightarrow$ C_5={{abcde}}
$\rightarrow$ prüfe ob {{abcd}{abce}{abde}{acde}{bcde}} in L_4

L_4={{abcd}, {abde}}
$\Rightarrow$ C_5={{abcde}}
$\rightarrow$ (prüfe ob {{abce}, ...} in L_4)

Datenstruktur für schnelle Untermengen-Operationen



Regeln finden

Feststellung:

für $A \mid \quad X : confidence((X - A) \Rightarrow A) = \frac{support(X)}{support(X - A)}$ und

für $B \mid \quad A : confidence((X - A) \Rightarrow A) \leq confidence((X - B) \Rightarrow B)$

Idee: Bilde zu $A_1 \subseteq A_2 \subseteq ... \subseteq A_{k-1} \subseteq X_k$ immer neue Obermengen, bis *minconf* unterschritten wird

Beispiel

X=abcde, *minconf* = 0.02

X-A	A	*confidence* (X-A ⇒ A)
abcd	e	0.03
abc	de	0.02
ab	cde	0.01 Abbruch!
a	bcde	
	a, b, c, d, ce, be, ae, bde, ade, ...	

$$confidence((X - A) \Rightarrow A) = \frac{support(X)}{support(X - A)}$$

ap-genrule Algorithmus

Initialisierung:
forall k-Artikelmengen X_k:
Erzeuge Co_1={Menge aller 1-elementigen Teilmengen von X_k}
Aufruf von ap-genrule(X_k, Co_1)
Regelerzeugung:
proc ap-genrule(X,Co_m)
forall $A \in Co_m$
conf=*confidence*(X-$A \Rightarrow A$)
if *conf* ≥ *minconf*
then output „Regel X-$A \Rightarrow A$, conf, support(X)“
else lösche A aus Co_m
Co_{m+1}= apriori-gen(Co_m)
ap-genrule(X, Co_{m+1})

AprioriTid

Feststellung: Nicht alle Transaktionen erzeugen häufige Artikelmengen.
Viele Transaktionen „bremsen“ den fortgeschrittenen Algorithmus

Definition: TID = Transaktionsnummer

Idee: Verwalte Menge C'_k von Elementen t = <*TID*, {X_k}>, so dass X_k potenziell häufige Artikelmengen sind
Einträge <*TID*, {}> müssen nicht aufgeführt werden

Datenbasis

TID	Artikel
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

C'_1

TID	Artikelgemenge
100	{ {1} {3} {4} }
200	{ {2} {3} {5} }
300	{ {1} {2} {3} {5} }
400	{ {2} {5} }

L_1

Artikelmenge	Sup
{1}	2
{2}	3
{3}	3
{5}	3

C_2

Artikelm.	Sup
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

C'_2

TID	Artikelgemenge
100	{ {1 3} }
200	{ {2 3} {2 5} {3 5} }
300	{ {1 2} {1 3} {1 5} {2 3} {2 5} {3 5}}
400	{ {2 5} }

L_2

Artikelm.	Sup
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

C_3

Artikelm.	Sup
{2 3 5}	2

C'_3

TID	Artikelgemenge
200	{ {2 3 5} }
300	{ {2 3 5} }

L_3

Artikelm.	Sup
{2 3 5}	2

AprioriTid-Algorithmus

Algorithmus: 1. Generiere alle häufigen 1-Itemsets = L_1

$C'_1 = D$

2. Generiere Kandidatenmenge C_k aus L_{k-1}
3. **forall** $t \in C'_{k-1}$

$C_t = \{c \in C_k \mid c[1]\ c[2] \ldots\ c[k-1] \in t.xk \ \wedge$

$c[1]\ c[2] \ldots c[k-2]\ c[k] \in t.xk\ \}$

5. **forall** $c \in C_t$

$c.count$ ++

6. **if** $C_t \neq \emptyset$ **then** C'_k += <$t.TID$, C_t>
7. $L_k = \{c \in C_k \mid c.count \geq mincount\}$
8. Gehe zu 2.

Welchen Algorithmus nehmen?

Pro AprioriTID

In späten Phasen wird nicht mehr die ganze Datenbank gescannt

C'_k passt evtl. in den Hauptspeicher

Pro Aproiori:

Einfach zu implementieren

Schlägt in vielen Fällen AprioriTID weil:

Contra AprioriTID

Overhead für C'_k in frühen Phasen sehr groß ($|C'_k| > |D|$)

Wenn $|C'_k|$>Hauptspeicher ⇒ Auslagern ⇒ langsam

⇒ Apriori-Hybrid

AprioriHybrid

Idee: Fange mit Apriori an,
wechsele zu AprioriTID, wenn C'_k
vermutlich in den Speicher passt

Schätze die Größe C'_k ab durch

$$\sum_{c \in C_k} support(c) + AnzahlTransaktionen$$ ab

Performance

Theoretisch schlecht:
$O(\|C\| \cdot |D|)$ ($\|C\|$=Summe der Größen aller Kandidaten)

$\|C\|$ exponentiell, weil alle Artikelmengen häufig sein können

In der Praxis aber: C_k wird sehr schnell klein

$\Rightarrow$ Apriori skaliert in der Praxis linear

Anwendung

Was heißt eigentlich „günstige“ Plazierung von Waren?

Reicht nicht *support(X)* aus?

Andere Anwendungen?

Einschränkungen? ($\neg a \Rightarrow$b)