

Vorlesungsplan

- 17.10. Einleitung
- 24.10. Ein- und Ausgabe
- 31.10. Reformationstag, Einfache Regeln
- 7.11. Naïve Bayes, Entscheidungsbäume
- 14.11. Entscheidungsregeln, Assoziationsregeln
- 21.11. Lineare Modelle, Instanzbasiertes Lernen
- 28.11. Clustering
- 5.12. Evaluation
- 12.12. Evaluation
- 19.12. Lineare Algebra für Data Mining
- 9.1. Statistik für Data Mining
- 16.1. Lineare Modelle, Support Vector Machines (SVM)
- 19.1. Vorlesung statt Übung: Bayes-Netze
- 23.1. Clustering
- 26.1. Vorlesung statt Übung: Clustering II
- 30.1. **Finden von häufigen Teilstrukturen**
- 2. 2. Klausur

Häufige Item-Mengen: die Schlüssel-Idee

- *Häufige Item-Menge*: Menge von Items mit minimaler Unterstützung (ist mind. in min-sup. Transaktionen enthalten)
 - *Monotonie-Prinzip*: Jede Teilmenge einer häufigen Item-Menge muß auch eine häufige Item-Menge sein
 - z.B., wenn $\{AB\}$ eine häufige Menge ist, müssen $\{A\}$ und $\{B\}$ auch häufig sein
 - Bestimme iterativ häufige Item-Mengen mit Kardinalität von 1 bis k (k -Item-Menge)

Apriori Algorithmus

- **Join Schritt:** C_k wird erzeugt durch Vereinigung von L_{k-1} mit sich selbst
- **Prune Schritt:** Jede $(k-1)$ -Item-Menge, die nicht häufig ist, kann nicht Teilmenge einer häufigen k -Item-Menge sein
- **Pseudo-code:**

C_k : Kandidaten für Item-Mengen der Größe k
 L_k : häufige Item-Mengen der Größe k

$L_1 = \{\text{häufige Items}\};$

for ($k = 1; L_k \neq \emptyset; k++$) **do begin**

C_{k+1} = erzeuge Kandidaten aus L_k ;

for each Transaktion t in der Datenbank **do**

 erhöhe den Zähler aller Kandidaten in C_{k+1} die in t enthalten sind

L_{k+1} = Kandidaten in C_{k+1} mit minimaler Unterstützung

end

return $\cup_k L_k$;

Methoden zur Verbesserung der Effizienz von Apriori

- **Hash-basiertes Zählen der Item-Mengen:** Eine k -Item-Menge, deren Hash-Bucket Zähler unter Min-Support liegt, kann nicht häufig sein
- **Transaktionsreduktion:** Eine Transaktion, die keine häufige k -Item-Menge enthält, ist nutzlos in folgenden DB Durchläufen
- **Partitionieren:** Jede Item-Menge, die potentiell häufig in der DB ist, muß in mindestens einer der Partitionen der DB häufig sein
- **Sampling:** Untersuche eine Teilmenge der gegebenen Daten, kleinerer min-support + eine Methode zum Prüfen der Vollständigkeit

Ist Apriori schnell genug? — Performanz-Engpässe

- Der Kern des Apriori Algorithmus:
 - Nutze häufige $(k - 1)$ -Item-Mengen zum Erzeugen der Kandidaten
 - Nutze DB Durchlauf zum Prüfen der Kandidaten
- Engpaß von *Apriori*: Kandidatenerzeugung
 - Riesige Kandidaten-Mengen:
 - 10^4 häufige 1-Item-Mengen erzeugen 10^7 Kandidaten 2-Item-Mengen
 - Zum Finden einer häufiges Item-Menge der Größe 100, müssen etwa $2^{100} \approx 10^{30}$ Kandidaten erzeugt werden.
 - Mehrfache Durchläufe über die DB:
 - Es werden $(n + 1)$ Durchläufe benötigt, n ist die Länge der größten Item-Menge

Finden von häufigen Item-Mengen ohne Kandidaten Erzeugung

- Komprimiere eine große DB in eine kompakten, Frequent-Pattern Baum (FP-tree)-Struktur
 - stark komprimiert, aber vollständige Information für das Finden von häufigen Item-Mengen
 - vermeidet DB Durchläufe
- FP-tree-basierte Methode zum Finden von häufigen Item-Mengen
 - Teile-und-Herrsche Prinzip: zerlege Gesamtaufgabe in kleinere Aufgaben
 - Vermeide Kandidatenerzeugung:
dafür Test auf einem Teil der DB

Konstruktion des FP-Baums aus den Transaction

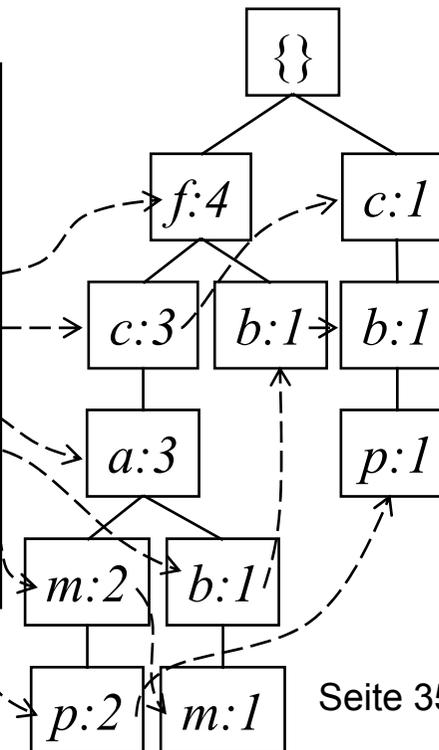
<i>TID</i>	<i>Items bought</i>	<i>(ordered) frequent items</i>
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

min_support = 3

Schritte:

1. Durchlaufe DB einmal, finde häufige 1-Itemmengen
2. Sortiere häufige Items absteigend nach Häufigkeit
3. Durchlauf DB nochmal, konstruiere FP-Baum

Zeiger Tabelle		
<i>Item</i>	<i>Häufig.</i>	<i>Zeiger</i>
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	



Vorteile der FP-Baum Struktur

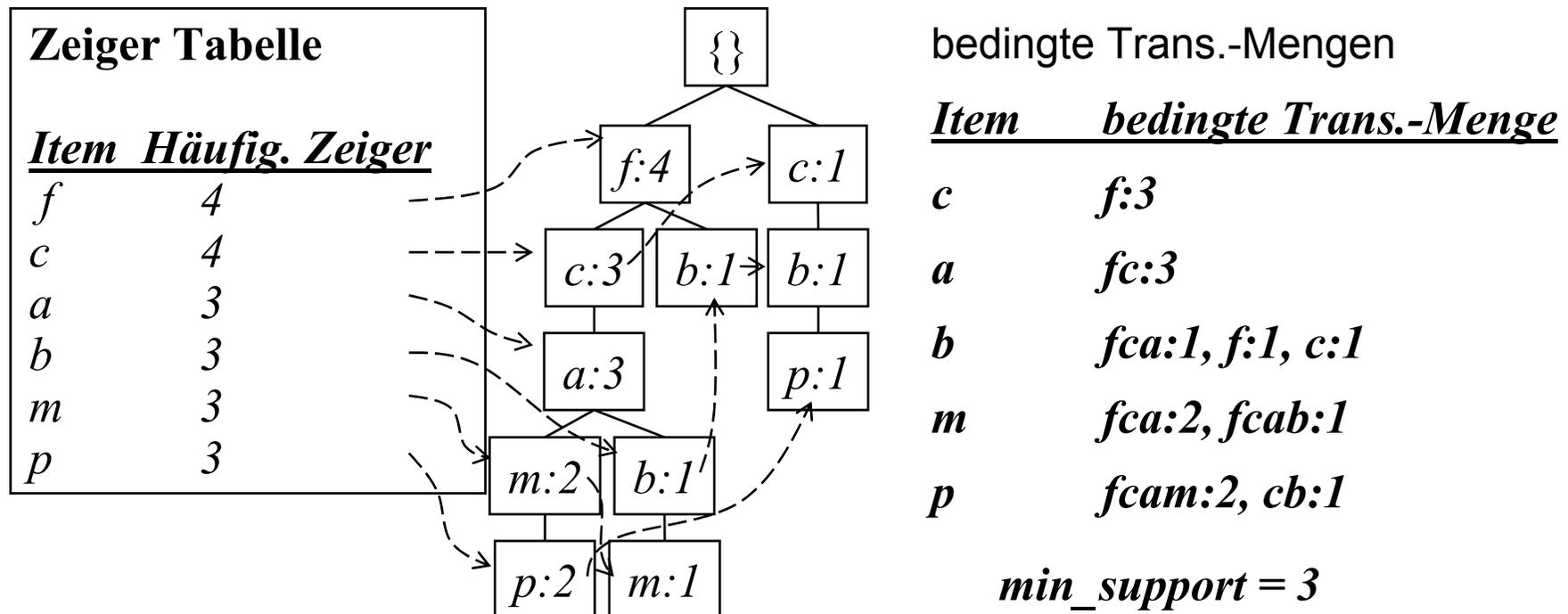
- Vollständigkeit:
 - Item-Mengen in Transaktionen werden nicht zerstört
- Kompaktheit
 - reduziert irrelevante Information — nicht-häufige Items werden gelöscht
 - Sortierung nach absteigender Häufigkeit: häufigere Items werden in vielen Transaktionen nur einmal repräsentiert
 - Beispiele für DBs mit Kompressionsrate 100

Schritte zum Verarbeiten der FP-Bäume

- 1) Konstruiere bedingte Transaktionsmenge für jeden Knoten in einem FP-Baum
- 2) Konstruiere bedingten FP-Bäume aus bedingten Transaktionsmengen
- 3) Durchsuche rekursiv die bedingten FP-Bäume und vergrößere die häufigen Item-Mengen, die bisher gefunden worden sind
 - Falls der bedingte FP-Baum nur einen Pfad enthält, erzeuge alle häufigen Item-Menge durch Aufzählen

Schritt 1: Vom FP-Baum zur bedingten Transaktionsmenge

- Beginne mit der Zeigertabelle und durchlaufe die Liste für jedes häufige Item
- Sammle alle Präfix-Pfade eines Item, um dessen bedingte Muster-Menge aufzubauen

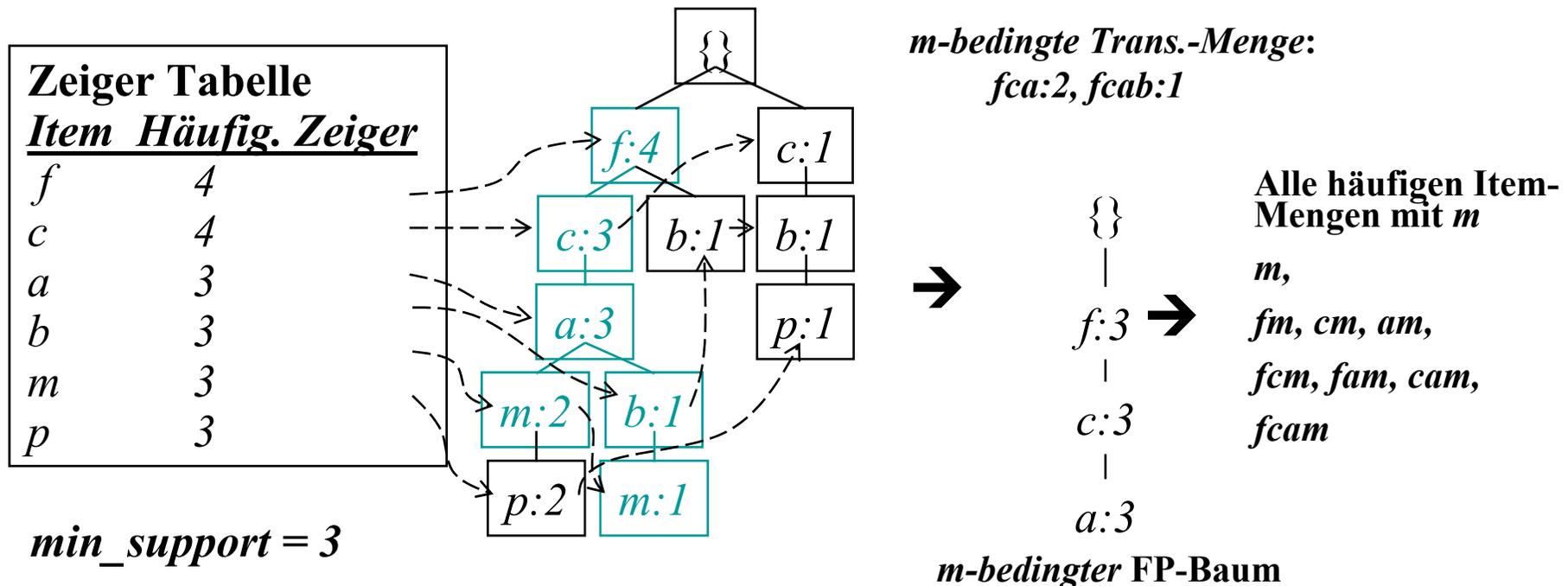


Eigenschaften von FP-Bäumen für die Konstruktion von bedingten Trans.-Mengen

- Eigenschaft der Item-Liste
 - Für jedes häufige Item a_i gilt, alle möglichen häufigen Item-Mengen werden beim Durchlaufen von a_i 's Liste erzeugt
- Präfix-Pfad Eigenschaft
 - Die Häufigkeit einer Item-Menge (repräsentiert durch den Präfix-Pfad P) für einen Knoten a_i wird im Zähler des Knotens a_i gespeichert.

Schritt 2: Konstruktion des bedingten FP-Baumes

- Für jede bedingte Trans. -Menge
 - Fasse die Zähler für jedes Item in der Menge zusammen
 - Konstruiere den FP-Baum für die häufigen Items dieser Menge



Finden von häufigen Item-Mengen durch Erzeugen von bedingten Trans.-Mengen

Item	Bedingte Trans.-Menge	Bedingter FP-Baum
p	$\{(fcam:2), (cb:1)\}$	$\{(c:3)\} p$
m	$\{(fca:2), (fcab:1)\}$	$\{(f:3, c:3, a:3)\} m$
b	$\{(fca:1), (f:1), (c:1)\}$	leer
a	$\{(fc:3)\}$	$\{(f:3, c:3)\} a$
c	$\{(f:3)\}$	$\{(f:3)\} c$
f	leer	leer

min_support = 3

Schritt 3: Rekursives Durchsuchen der bedingten FP-Bäume

min_support = 3

$\{\}$
 $|$
 $f:3$
 $|$
 $c:3$
 $|$
 $a:3$

m-bedingter FP-Baum

Bedingte Trans.-Menge für "am": (fc:3)

$\{\}$
 $|$
 $f:3$
 $|$
 $c:3$

am-bedingter FP-Baum

Bedingte Trans.- Menge für "cm": (f:3)

$\{\}$
 $|$
 $f:3$

cm-bedingter FP-Baum

Bedingte Trans.- Menge für "cam": (f:3)

$\{\}$
 $|$
 $f:3$

cam-bedingter FP-Baum

FP-Baum mit nur einem Pfad

- Angenommen ein FP-Baum T hat nur einen Pfad P
- Die vollständige Menge der häufigen Item-Mengen von T kann durch Aufzählen aller Kombinationen der Teilpfade von P erzeugt werden

$min_support = 3$

$\{\}$
|
 $f:3$
|
 $c:3$
|
 $a:3$



Alle häufigen Item-Mengen mit m

$m,$

$fm, cm, am,$

$fcm, fam, cam,$

$fcam$

m -bedingter FP-Baum

Prinzip des Vergrößerns der häufigen Item-Mengen

- Eigenschaft
 - Sei α eine häufige Item-Menge in der DB, B α 's bedingte Trans.-Menge und β eine Item-Menge in B . Dann ist $\alpha \cup \beta$ eine häufige Item-Menge in der DB, gdw. β ist häufig in B .
- “*abcdef*” ist eine häufige Item-Menge, gdw.
 - “*abcde*” ist eine häufige Item-Menge, und
 - “*f*” ist häufig in der Menge der Transaktionen die “*abcde*” enthalten

Warum ist das Vergrößern häufiger Item-Mengen schnell?

- Experimentelle Studie zeigt
 - FP-growth ist deutlich schneller als Apriori
- Erklärung
 - Keine Erzeugung von Kandidaten, kein Kandidaten-Test
 - Kompakte Datenstruktur
 - Vermeidung von wiederholten DB Durchläufen
 - Basisoperationen sind Zählen und FP-Baum Konstruktion

FP-growth vs. Apriori: Skalierbarkeit bezüglich Min-Support

