Übersicht

- Grundlagen für Assoziationsregeln
- Apriori Algorithmus
- Verschiedene Datenformate
- Finden von Assoziationsregeln mit mehren unteren Schranken für Unterstützung
- Finden von Assoziationsregeln für Klassifikation
- Finden von sequenziellen Mustern
- Zusammenfassung

Probleme von Assoziationsregeln

- Einzelne untere Schranke für Unterstützung: implizite Annahme, dass alle Artikel in den Daten vergleichbar sind und ähnliche Häufigkeiten haben.
- Leider falsch: In vielen Anwendungen sind einige Artikel sehr häufig, während andere eher selten sind.

  Z.B., Supermarktdaten, *Küchenmaschinen oder Pfannen* werden wesentlich seltener als *Brot* und *Milch* gekauft.

Problem mit seltenen Artikeln

- Wenn die Häufigkeit von Artikeln stark variiert, treten zwei Probleme auf
  - Wenn die untere Schranke für Unterstützung zu hoch ist, werden Regeln mit seltenen Artikeln von vorherein nicht gefunden.
  - Um solche Artikel mit einzubeziehen, muß die untere Schranke sehr niedrig sein. Dies führt zu einer kombinatorischen Explosion, weil die häufigen Artikel miteinander in allen möglichen Kombinationen gefunden werden.

Modell für mehrere untere Schranken für Unterstützung

- Die minimale Unterstützung einer Regel wird durch mehrere untere Schranken für die in der Regel auftretenden Artikel *(minimum item supports,* MIS) festgelegt.
- Jeder Artikel a kann eine eigene untere Schranke MIS(a) haben.
- Durch eine Menge von MIS-Werten für verschiedene Artikel können verschiedene untere Schranken für Unterstützung für verschiedene Regeln festgelegt werden.
- Um zu vermeiden, dass sehr häufige und sehr seltene Artikel in einer Artikelmenge gemeinsam auftreten wird noch die **Unterstützungsdifferenz-Bedingung** eingeführt.

$max_{i \in s}\{sup(i)\} - min_{i \in s}\{sup(i)\} \leq \varphi$

Untere Schranke für Unterstützung für eine Regel

- Sei MIS(i) der MIS-Wert eines Artikels i. Die untere Schranke einer Regel R ist der kleinste MIS-Wert der Artikel in der Regel.
- Regel R: $a_1, a_2, \ldots, a_k \rightarrow a_{k+1}, \ldots, a_r$ erfüllt die untere Schranke, wenn ihre Unterstützung $\geq$

  $\min(\text{MIS}(a_1), \text{MIS}(a_2), \ldots, \text{MIS}(a_r))$.

Beispiel

- Folgende Artikel:

  bread, *shoes*, *clothes*

  Die von Anwender vorgegeben MIS-Werte:

  MIS(*bread*) = 2% MIS(*shoes*) = 0.1%

  MIS(*clothes*) = 0.2%

  Die folgende Regel erfüllt nicht ihre untere Schranke:

  clothes $\rightarrow$ *bread* [sup=0.15%,conf =70%]

  Die folgende Regel erfüllt ihre untere Schranke :

  clothes $\rightarrow$ *shoes* [sup=0.15%,conf =70%]

Monotonie Eigenschaft

- In dem neuen Modell gilt die Monotonie Eigenschaft nicht mehr (?)

z.B., betrachte vier Artikel 1, 2, 3 und 4 in einer Datenbank. Ihre MIS-Werte sind

MIS(1) = 10% MIS(2) = 20%

MIS(3) = 5% MIS(4) = 6%

{1, 2} mit angenommener Unterstützung 9% ist nicht häufig, aber {1, 2, 3} und {1, 2, 4} könnten trotzdem noch häufig sein.

Lösungs des Problems

- Ordne alle Artikel in *I* nach ihren MIS-Werten (totale Ordnung).
- Diese Ordnung wird implizit im ganzen Algorithmus bei jeder Artikelmenge genutzt.
- Jede Artikelmenge *w* hat folgende Form:

  {*w*[1], *w*[2], …, *w*[*k*]}, besteht aus den Artikeln,

  w[1], *w*[2], …, *w*[*k*],

  wobei MIS(*w*[1]) ≤ MIS(*w*[2]) ≤ … ≤ MIS(*w*[*k*]).

Der MSapriori Algorithmus

Algorithm MSapriori(*T, MS,* φ) // φ ist für die Unterstützungsdifferenz-Bedingung

$M \leftarrow$ sort(*I, MS*);
$L \leftarrow$ init-pass(M, T);
$F_1 \leftarrow \{\{i\} \mid i \in L, i.count/n \geq \text{MIS}(i)\}$;
for (k = 2; $F_{k-1} \neq \emptyset$; k++) **do**
  if k=2 **then**
    $C_k \leftarrow$ level2-candidate-gen(L, φ)
  else $C_k \leftarrow$ MScandidate-gen(F_{k-1}, φ);
  end;
  for each transaction $t \in T$ **do**
    for each candidate $c \in C_k$ **do**
      if c is contained in t **then**
        $c.count$++;
      if $c - \{c[1]\}$ is contained in t **then**
        $c.tailCount$++
    end
  end
  $F_k \leftarrow \{c \in C_k \mid c.count/n \geq MIS(c[1])\}$
end
return $F \leftarrow \bigcup_k F_k$;

Kandidaten Erzeugung

- Spezielle Behandlung wird gebraucht:
  - Sortiere die Artikel bezüglich ihrer MIS-Werte
  - Erster Durchlauf über die Daten (ersten drei Zeilen)
    - Diskutieren wir in der jetzt in der Vorlesung.
  - Kandidatenerzeugung auf Ebene 2
    - Lesen Sie als Übung zu Hause.
  - Pruning Schritt auf Ebene k ($k > 2$) und Kandidaten Erzeugung.
    - Lesen Sie als Übung zu Hause.

Erster Durchlauf über die Daten

- Ein Durchlauf wird gebraucht, um die Häufigkeit aller Artikel zu bestimmen.
- Folge dann der Ordnung, um den ersten Artikel *i* zu finden, der seine untere Schranke erfüllt MIS(*i*).
  - *i* wird in *L* eingefügt.
  - Für alle nachfolgenden Artikel *j* in *M* nach *i*: falls *j.count* / *n* ≥ MIS(*i*), dann wird *j* auch *L* eingefügt, wobei *j.count* die Unterstützung von *j* ist und *n* die Anzahl der Transaktion in *T*. Warum?
- *L* wird in der Funktion level2-candidate-gen gebraucht.

Erster Durchlauf über die Daten: Beispiel

- Betrachte vier Artikel 1, 2, 3 und 4 in einer DB. Ihre MIS-Werte sind:

  MIS(1) = 10% MIS(2) = 20%

  MIS(3) = 5% MIS(4) = 6%

- Angenommen es gibt 100 Transaktionen. Der erste Durchlauf erzeugt die folgenden Unterstützungen:

  {3}.*count* = 6, {4}.*count* = 3,

  {1}.*count* = 9, {2}.*count* = 25.

- **Dann** L = {3, 1, 2}, und F_1 = {{3}, {2}}
- Artikel 4 ist nicht in L weil 4.*count* / n < MIS(3) (= 5%),
- {1} ist nicht in F_1 weil 1.*count* / n < MIS(1) (= 10%).

level2-candidate-gen

```
Function level2-candidate-gen(L,phi)
    C2 <- empty set
    for each item l in L in the same order do
        if l.count / n >= MIS(l) then
            for each item h in L that is after l do
                if h.count/n >= MIS(l) and |sup(h)-sup(l)| <= phi
                    C2 <- C2 union {{l,h}} // füge {l,h} in C2 ein
```

Beispiel

- Betrachte vier Artikel 1, 2, 3 und 4 in einer DB. Ihre MIS-Werte sind:

  MIS(1) = 10% MIS(2) = 20%

  MIS(3) = 5% MIS(4) = 6%
- 100 Transaktionen, Unterstützungen:

  {3}.*count* = 6, {4}.*count* = 3,

  {1}.*count* = 9, {2}.*count* = 25.
- **Dann** L = {3, 1, 2}, und F_1 = {{3}, {2}}
- C2 = {{3,1}}
  - {1,2} ist kein Kandidat weil {1}.*count* < *MIS(1)*
  - *{3,2}* ist kein Kandidat weil |{3}.*count* - {2}.*count*| > *10*

MScandidate-gen

```
Function MScandidate-gen(Fk-1,phi)
    forall f1,f2 in Fk
        with f1={i1,...,ik-2,ik-1}
        and f2={i1,...,ik-2,i'k-1}
        and ik-1<i'k-1 and |sup(ik-1)-sup(i'k-1)| <=phi do
            c <- {i1,...,ik-1,i'k-1}
            Ck <- Ck union {c}
            for each (k-1)-subset s of c do
                if(c[1] in s) or (MIS(c[1])==MIS(c[2]) then
                    if(s not in Fk-1) then
                        delete c from Ck
            endfor
    endfor
    return Ck
```

Beispiel

- Sei F3={{1,2,3},{1,2,5},{1,3,4},{1,3,5},{1,4,5},{1,4,6}, {2,3,5}}
- Join Schritt erzeugt (ohne sup-Bedingung)
  - {1,2,3,5}, {1,3,4,5} und {1,4,5,6}
- Prune Schritt löscht
  - {1,4,5,6} weil {1,5,6} nicht in F3
- {1,3,4,5} wird nicht gelöscht, trotzdessen {3,4,5} nicht in F3 ist, weil MIS(3)>MIS(1)
  - Falls MIS(3)==MIS(1) wäre, könnte {1,3,4,5} gelöscht werden
- C4 = {{1,2,3,5},{1,3,4,5}}

Regel Erzeugung

- Die nächsten beiden Zeilen im MSapriori Algorithmus sind für die Regelerzeugung wichtig, sie werden für den Apriori algorithm nicht gebraucht

  if c – {c[1]} ist in t **then**

  $c.tailCount$++

- Viele Regeln können ohne diesen Zähler nicht erzeugt werden.
- Warum?

Zusammenfassung

- Das Modell mit mehreren unteren Schranken enthält das Modell mit einer unteren Schranke
- Es ist ein realistisches Modell für praktische Anwendungen.
- Das Modell kann auch Regeln mit seltenen Artikeln finden, ohne viele sinnlose Regeln zu generieren.
- Wenn MIS-Werte von einigen Artikeln auf 100% (oder mehr) gesetzt sind, werden Regeln erzeugt, die nicht nur diese Artikel enthalten.

Übersicht

- Grundlagen für Assoziationsregeln
- Apriori Algorithmus
- Verschiedene Datenformate
- Finden von Assoziationsregeln mit mehren unteren Schranken für Unterstützung
- Finden von Assoziationsregeln für Klassifikation
- Finden von sequenziellen Mustern
- Zusammenfassung

Finden von Assoziationsregeln für Klassifikation

- Normale Assoziationsregeln haben keine Zielartikel auf der rechten Seite.
- Es werden alle möglichen Regeln gefunden, die in den Daten existieren, d.h. jeder Artikel kann auf der rechten Seite auftauchen.
- In einigen Anwendungen sind Zielartikel schon vorgegeben.
  - z.B, Menge von Textdokumenten von denen die Themen bekannt sind. Frage: Welche Wörter sind mit welchen Themen korreliert?

Problem-Definition

- Sei T eine Menge von n Transaktionen.
- Jede Transaktion ist auch mit einer Klasse y gekennzeichnet.
- Sei I die Menge aller Artikel in T und Y die Menge der Klassen und $I \cap Y = \emptyset$.
- Eine **Klassen-Assoziationsregel** (**CAR**) ist eine Implikation der Form

  $X \rightarrow y$, mit $X \subseteq I$, und $y \in Y$.
- Die Definitionen von **Unterstützung** und **Konfidenz** sind die gleichen wie für normale Assoziationsregeln.

Beispiel

- **Textdokumente**

doc 1:	Student, Teach, School	: Education
doc 2:	Student, School	: Education
doc 3:	Teach, School, City, Game	: Education
doc 4:	Baseball, Basketball	: Sport
doc 5:	Basketball, Player, Spectator	: Sport
doc 6:	Baseball, Coach, Game, Team	: Sport
doc 7:	Basketball, Team, City, Game	: Sport

- Sei *minsup* = 20% und *minconf* = 60%. Zwei Beispiele für Klassen-Assoziationsregeln:

  Student, School $\rightarrow$ Education [sup= 2/7, conf = 2/2]

  game $\rightarrow$ Sport [sup= 2/7, conf = 2/3]

Algorithmus

- Im Gegensatz zu normalen Assoziationsregeln, CARs können direkt in einem Schritt gefunden werden.
- Die Hauptoperation ist alle Artikelmengen einer Klasse zu finden, deren Unterstützung die Mindestschranke *minsup* überschreitet. Eine Artikelmenge einer Klasse hat die Form:

  (*condset*, *y*)

  wobei **condset** eine Menge von Artikeln aus *I* ist (d.h. $condset \subseteq I$), und $y \in Y$ ist eine Klasse.
- Jede Artikelmenge einer Klasse repräsentiert die Regel:

  $condset \rightarrow y$,
- Der Apriori-Algorithmus kann leicht geändert werden um alle CARs zu erzeugen. Wie?

Ändern von Apriory für CARs

Algorithm Apriori(*T*)

$C_1 \leftarrow$ init-pass(T);
$F_1 \leftarrow \{f \mid f \in C_1, f.\text{count}/n \geq minsup\}$; // n: no. of transactions in T
for (k = 2; $F_{k-1} \neq \emptyset$; k++) **do**
  $C_k \leftarrow$ candidate-gen(F_{k-1});
  for each transaction $t \in T$ **do**
    for each candidate $c \in C_k$ **do**
      if c is contained in t **then**
        $c.count$++;
    end
  end
  $F_k \leftarrow \{c \in C_k \mid c.count/n \geq minsup\}$
end
return $F \leftarrow \bigcup_k F_k$;

Mehrere untere Schranken für Klassenunterstützung

- Mehrere untere Schranken für Unterstützung können auch hier angewendet werden.
- Jeder Klasse kann eine andere untere Schranke gegeben werden.
- Zu Beispiel ein Zweiklassenproblem mit den Klassen Yes und No
  - Regeln für Klasse Yes sollen eine minimale Schranke von 5%
  - Regeln für Klasse No sollen eine minimale Schranke von 10% haben
- Wenn die untere Schranke für eine Klasse über 100% gesetzt wird, zwingen wir den Algorithmus keine Regeln für diese Klasse zu erzeugen.
  - Nützliche Anwendungen für diesen Trick?