5. Klassifikations- und Assoziationsregeln

Voraussetzung weiterhin:
Tabelle von Instanzen mit mehrwertigen Attributen
("mehrwertiger Kontext" im Sinne der Begriffsanalyse)

eines der Attribute binär und besonders ausgezeichnet – soll aus den anderen Attributen erschlossen werden (Lernproblem)

Klassifikationsregeln: haben die Form

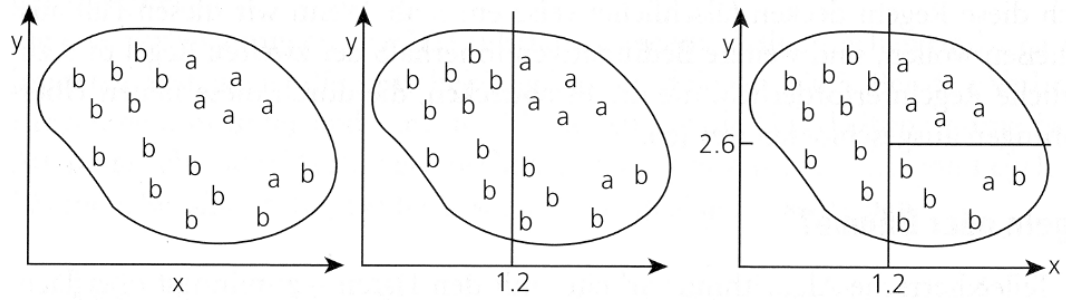
IF (Konjunktion von speziellen Attributbelegungen)
THEN Zielattribut = yes *(oder no)*

beim Entscheidungsbaum-Lernen:

- komplette Trainingsmenge wird herangezogen (in jedem Schritt)
- schrittweise Verfeinerung des Entscheidungsbaumes (bzw. der entsprechenden disjunktiven Verkettung von Regeln)

jetzt:
- in jedem Schritt wird eine Regel gelernt
- die Instanzen, die dadurch erfasst werden, werden entfernt
- dieser Vorgang wird für die Menge der restlichen Instanzen wiederholt...

Beispiel mit 2 numerischen Attributen:



Ergebnis: System von *Klassifikationsregeln* (disjunktiv verknüpft)

Abdeckungsalgorithmus (*sequential covering*)

vorausgesetzter Unter-Algorithmus: LEARN-ONE-RULE
liefert einzelne Regel, die nur eine Teilmenge der Trainingsmenge abdeckt, dort aber möglichst korrekt ist

SEQUENTIAL-COVERING(*Target_attribute*, *Attributes*, *Examples*, *Threshold*)

- *Learned_rules* ← {}
- *Rule* ← LEARN-ONE-RULE(*Target_attribute*, *Attributes*, *Examples*)
- while PERFORMANCE(*Rule*, *Examples*) > *Threshold*, do
  - *Learned_rules* ← *Learned_rules* + *Rule*
  - *Examples* ← *Examples* − {examples correctly classified by *Rule*}
  - *Rule* ← LEARN-ONE-RULE(*Target_attribute*, *Attributes*, *Examples*)
- *Learned_rules* ← sort *Learned_rules* accord to PERFORMANCE over *Examples*
- return *Learned_rules*

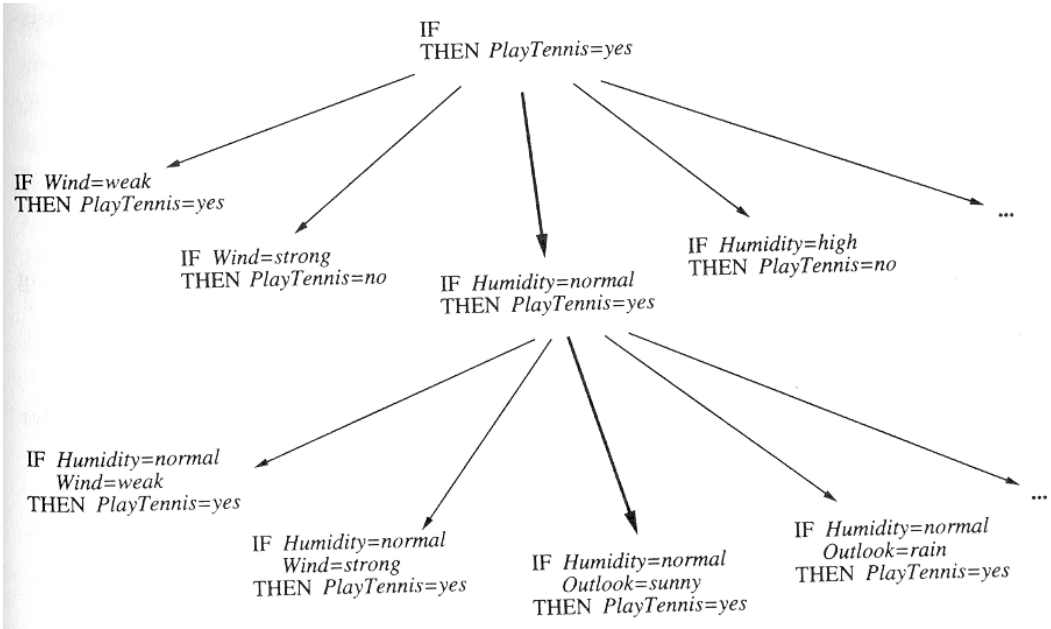
(aus Mitchell 1997)

Greedy-Algorithmus ⇒ nicht garantiert, dass kleinste oder beste Regelmenge gefunden wird

für Subroutine LEARN-ONE-RULE:
verschiedene Ansätze möglich

wie beim Entscheidungsbaum-Lernen vorgehen (ID3), aber in jedem Schritt nur dem "besten" Zweig folgen:

- starte mit der allgemeinsten Regelbedingung (leere Voraussetzung, immer erfüllt)
- füge denjenigen Attribut-Test hinzu, der die Klassifikationsleistung der Regel auf der jeweiligen Trainingsmenge am meisten erhöht (z.B. nach Informationsgewinn-Kriterium)
- wiederhole dies



(Mitchell 1997)

Modifikation: "**Strahlsuche**" (*beam search*)

- statt in jedem Schritt nur 1 Attribut zu betrachten, speichere jedesmal Liste der k besten Kandidaten-Attribute
- Nachfolger werden für jeden dieser Kandidaten bestimmt
- resultierende Menge wird erneut auf die k besten reduziert

Verwendung im Algorithmus CN2 (Clark & Niblett 1989)

LERN-EINE-REGEL(*Ziel_Attribut*, *Attribute*, *Beispiele*, *k*)

Gibt eine einzelne Regel zurück, die einige der Beispiele abdeckt. Benutzt eine General-to-Specific Baumsuche um die beste Regel zu finden, geführt von der PERFORMANCE Metrik.

- Initialisiere *Beste_Hypothese* mit der allgemeinsten Hypothese ∅
- Initialisiere *Kandidaten_Hypothesen* mit der Menge {*Beste_Hypothese*}
- While *Kandidaten_Hypothesen* nicht leer, Do
  1. Generiere die am meisten spezifische nächste *Kandidaten_Hypothesen*
     - *Alle_Einschränkungen* ← die Menge von allen Einschränkungen der Form ($a = v$), wobei a ein Mitglied von *Attribute* ist und v ein Wert von a, der in der aktuellen Menge von *Beispiele* vorkommt
     - *Neue_Kandidaten_Hypothesen* ←
       Für jedes *h* aus *Kandidaten_Hypothesen*,
       Für jedes *c* aus *Alle_Einschränkungen*,
       - Kreiere eine Spezialisierung von *h* indem eine Einschränkung *c* hinzugefügt wird
     - Entferne alle doppelten, widersprüchlichen und nicht maximal spezifischen Hypothesen aus *Neue_Kandidaten_Hypothesen*
  2. Aktualisiere *Beste_Hypothese*
     - Für alle h aus *Neue_Kandidaten_Hypothesen* do
       - Wenn(PERFORMANCE(*h*,*Beispiele*,*Ziel_Attribut*) > PERFORMANCE(*Beste_Hypothese*,*Beispiele*,*Ziel_Attribut*))
         Dann *Beste_Hypothese* ← *h*
  3. Aktualisiere *Kandidaten_Hypothesen*
     - *Kandidaten_Hypothesen* ← die k besten Mitglieder von *Neue_Kandidaten_Hypothesen*, bezüglich der PERFORMANCE
- Return eine Regel von der Form
  „WENN *Beste_Hypothese* DANN *Vorhersage*“ wobei, *Vorhersage* der häufigste Wert aus *Ziel_Attribut* über *Beispiele* ist, die zu *Beste_Hypothese* passen.

PERFORMANCE(*h*, *Beispiele*, *Ziel_Attribut*)

- *h_Beispiele* ← die Untermenge von *Beispiele*, die zu *h* passt
- return *Entropy*(*h_Beispiele*), wo die Entropy die *Ziel_Attribut* respektiert

(Scheffer & Bickel 2004)

- der Algorithmus arbeitet "general to specific"
- gefundene Regel mit größter Klassifikationsleistung wird ausgegeben – nicht notwendigerweise die zuletzt erzeugte
- kann immer noch suboptimale Regel liefern
- dies ist i.allg. nicht schlimm, da Aufruf im Abdeckungs-Algorithmus mehrfach erfolgt

Unterschied zum Entscheidungsbaum-Lernen:

- beim *sequential covering* wird zwischen (Attribut, Wert)-Paaren verglichen – durch Vergleich der dadurch abgedeckten Teilmengen von Trainingsbeispielen
- beim Entscheidungsbaum-Lernen wird zwischen Attributen verglichen – durch Vergleich der Partitionen der Gesamt-Trainingsmenge, die durch das Attribut generiert wird

⇒ bei *sequential covering* größere Zahl von unabhängigen Entscheidungen erforderlich
⇒ anwendbar eher bei großen Trainingsdatensätzen

Wie beim Entscheidungsbaum-Lernen besteht Gefahr des Overfitting
⇒ evtl. Postpruning der Regeln erforderlich (Entfernen von Voraussetzungen von Regeln)

es gibt auch *specific-to-general* Regelfindungsalgorithmen:

- beginne mit Zufallsauswahl von Positiv-Instanzen
- verallgemeinere die daraus konstruierte spezielle Regel durch Hinzunahme weiterer Trainingsbeispiele

(GOLEM, Muggleton & Feng 1990)
Anwendungen:

- Vorhersage von Proteinstrukturen aus Sequenzinformation
- Vorhersage des Nutzens von Medikamenten basierend auf Molekülstruktur
- Veröffentlichung der gefundenen Regeln in wissenschaftlichen Zeitschriften

bisher betrachtete Regeln enthalten als Attributwerte nur Konstanten (*propositional rules*).

Zulassen von Variablen:
Regeln der Prädikatenlogik 1. Ordnung ("*first-order rules*")

⇒ **Inductive Logic Programming** (**ILP**):
Lernen von solchen Regeln

spezielle Form von Regeln 1. Ordn.: *Horn-Klauseln*
Programme in PROLOG bestehen aus Horn-Klauseln.

Terme: z.B. Fritz, *x*, *age*(Fritz), *age*(*x*) (*age* ist Funktion)

Prädikate: so heißen jetzt die Attribute, z.B. Female(Mary)

Literal: ein Prädikat oder seine Negation, angewandt auf eine Liste von Termen

Klausel: Disjunktion von endlich vielen Literalen
(darin enthaltene Variablen werden implizit als mit dem Allquantor $\forall$ versehen angenommen)

Horn-Klausel: Ausdruck der Form $H \leftarrow (L_1 \wedge ... \wedge L_n)$
darin sind H, L_1, ..., L_n Literale ohne Negation.
Dieser Ausdruck ist gleichwertig zu $H \vee \neg L_1 \vee ... \vee \neg L_n$.
In PROLOG: H :- L_1 , ..., L_n .
H: Kopf, L_1, ..., L_n : Rumpf der Regel.

Beispiel für den Nutzen der Verwendung von Variablen:

Datensätze der Form

$$\begin{array}{lll} \langle Name_1 = Sharon, & Mother_1 = Louise, & Father_1 = Bob, \\ Male_1 = False, & Female_1 = True, & \\ Name_2 = Bob, & Mother_2 = Nora, & Father_2 = Victor, \\ Male_2 = True, & Female_2 = False, & Daughter_{1,2} = True \rangle \end{array}$$

(Zielattribut *Daughter*$_{1,2}$)

Anwendung von Regel-Lernen ohne Variablen würde auf sehr spezielle Regeln führen, z.B.

$$\begin{array}{ll} \text{IF} & (Father_1 = Bob) \wedge (Name_2 = Bob) \wedge (Female_1 = True) \\ \text{THEN} & Daughter_{1,2} = True \end{array}$$

(für praktische Zwecke völlig uninteressant)

Interessant dagegen: Regeln wie

$$\text{IF} \quad Father(y, x) \wedge Female(y), \quad \text{THEN} \quad Daughter(x, y)$$

wobei *x* und *y* Variablen sind, die für Personennamen stehen.

Algorithmus FOIL (*First-Order Inductive Logic*; Quinlan 1990) zum Lernen von Hornklauseln

- im wesentlichen eine Erweiterung des propositionalen Sequential Covering
- die gelernten Hypothesen: "fast" Mengen von Hornklauseln
- die Literale in diesen Klauseln enthalten keine Funktionen
- die Literale im Rumpf der Regeln dürfen auch negiert sein

erfolgreiche Anwendungen (u.a.):

- Klassifikation von Schachpositionen (erlaubt vs. verboten)
- Lernen einer rekursiven Definition von Quicksort

FOIL(*Ziel_Prädikat*, *Prädikate*, *Beispiele*)

- *Pos* ←die Beispiele, bei denen das *Ziel_Prädikat* wahr ist
- *Neg* ←die Beispiele, bei denen das *Ziel_Prädikat* falsch ist
- *Gelernte_Regeln* ← { }
- While *Pos*, Do

  Lerne eine *NeueRegel*
  - *NeueRegel* ←die Regel, die *Ziel_Prädikat* ohne Vorbedingungen erfüllt
  - *NeueRegelNeg* ← *Neg*
  - While *NeueRegelNeg*, Do

    Ergänze ein neues Literal, um *NeueRegel* zu spezialisieren
    - *KandidatenLiterale* ← erstelle neue Kandidaten Literale für *NeueRegel*, basierend auf *Prädikate*
    - *BestesLiteral* ← argmax *Foil_Gain*(*L*, *NeueRegel*)
    - Ergänze *BestesLiteral* zu den Vorbedingungen von *NeueRegel*
    - *NeueRegelNeg* ← Untermenge von *NeueRegelNeg*, die die Vorbedingungen von *NeueRegel* erfüllt
  - *Gelernte_Regeln* ← *Gelernte_Regeln* + *NeueRegel*
  - *Pos* ← *Pos* – {Mitglieder von *Pos*, die von *NeueRegel* abgedeckt werden}
- return *Gelernte_Regeln*

(Scheffer & Bickel 2004)

- äußere Schleife lernt neue Regeln
- entfernt die positiven Trainingsbeispiele, die von der letzten Regel abgedeckt werden
- innere Schleife: ähnelt Erweiterung von LEARN-ONE-RULE

jedoch:

- FOIL sucht nur nach Regeln, die voraussagen, dass das Ziel-Literal TRUE ist
- FOIL vollführt einfaches hillclimbing (keine Strahlsuche, bzw. nur Strahlsuche mit Breite 1)
- andere Schritte zur Kandidatengenerierung für die Regel
- anderes Maß für die Klassifikationsleistung: insbesondere müssen verschiedene Variablenbelegungen ("Bindungen") berücksichtigt werden

Kandidatengenerierung in FOIL:

Man betrachte die Regel $P(x_1, ..., x_n) \leftarrow L_1 ... L_n$, wobei $L_1 ... L_n$ Literale für die Vorbedingungen sind und $P(x_1, ..., x_n)$ ist der Regelkopf. FOIL kreiert einen spezielleren Kandidaten dieser Regel indem es neue Literale L_{n+1} betrachtet, die eine der folgenden Bedingungen erfüllen:

- $Q(v_1, ..., v_r)$, wobei Q ein beliebiger Name ist für ein Prädikat, das in *Prädikate* vorkommt und die v_i entweder neue Variablen sind oder Variablen, die bereits in der Regel vorkommen. Mindestens ein v_i muss bereits in der Regel existieren.
- *Equal*(x_j, x_k), wobei x_j und x_k Variablen sind, die bereits in der Regel vorkommen.
- Die Negation von einer der oberen Formen von Literalen.

Bewertungsfunktion für die Steuerung der Suche:

- beruht auf Anzahl der positiven und negativen Bindungen (Variablenbelegungsmöglichkeiten) vor und nach der Erweiterung der Regel durch ein neues Literal
- betrachte Regel R und ein Kandidaten-Literal L, das evtl. zum Rumpf von R hinzugefügt werden soll
- sei R' die Regel, die entsteht, wenn man L zu R hinzufügt

Bewertungsfunktion:

$$\text{Foil\_Gain}(L, R) \equiv t\left(\log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right), \quad (1)$$

wobei p_0 die Anzahl von positiven Bindungen der Regel R, n_0 die Anzahl von negativen Bindungen von R, p_1 die Anzahl von positiven Bindungen von der Regel R' und n_1 die Anzahl von negativen Bindungen von R' ist. Außerdem ist t die Anzahl von positiven Bindungen der Regel R, die nach dem Hinzufügen von L zu R weiterhin abgedeckt werden. Wird eine neue Variable durch L eingeführt, dann wird jede Originalbindung so lange als abgedeckt betrachtet, wie einige von den Bindungen, die sie erweitern, in den Bindungen von R' vorkommen.

(Marquardt, Scheffer & Bickel 2004)

FOIL - Diskussion und Erweiterungen

+ Übertragung effizienter Meth. zum Entscheidungsbaumlernen auf Lernen von Hornformeln
+ kann rekursive Definitionen lernen
+ Ausnutzung von Hintergrundwissen (allerdings nur Fakten)
- Einschränkung auf funktionsfreie Hornlogik
- kann keine neuen Prädikate erfinden
- benötigt positive und negative Beispiele zum Lernen
- heuristisches (kurzsichtiges) Hill-Climbing
- trotz Hill-Climbing hohe Komplexität (Vielzahl möglicher neuer Rumpfliterale)

Erweiterungen, z.B.

- Suche mit Backtracking
- Typisierte Prädikate -> Reduktion der Komplexität (FOCL)
- Auch Regeln im Hintergrundwissen (FOCL)
- Vorgabe von partiellen Klauseln (FOCL)

Anderer Ansatz zum Lernen von Regeln mit Variablen (first-order rules):

- Umkehrung der logischen Deduktion
- Deduktion formalisierbar / automatisierbar z.B. durch *Resolution* als standardisierter logischer Schluss (vgl. PROLOG)
- Umkehrung der Resolution: Basis von CIGOL (Muggleton & Buntine 1988).

siehe dazu Mitchell (1997).

Vorteile der Induktiven Logik-Programmierung allgemein:

- Lernen unter Einbezug von Hintergrundwissen
- Lernen relationalen Wissens, das nicht mit Attribut-Wert-Strukturen beschreibbar ist
- Theoretische Fundierung (Logik)
- Mächtige Lernverfahren, z.T. genügen sehr wenige Beispiele
- Erfinden von Prädikaten

(Jung 2003)

Lernen von Assoziationsregeln

jetzt: kein spezielles Zielattribut in den Daten

es sollen Zusammenhänge zwischen *irgendwelchen* Attributen automatisch gefunden werden: "Assoziationsregeln"

- Verfahren zum Entdecken von Assoziationsregeln sind typische Data Mining Methoden.
- Assoziationsregeln beschreiben Zusammenhängen zwischen gemeinsam auftretenden Merkmalsausprägungen.
- Algorithmen zur Bestimmung von Assoziationsregeln sind unüberwachte Lernverfahren, d.h. ohne Lehrersignale.
- Typisches Anwendungsfeld: Zusammenhänge beim Einkauf, die sogenannte Warenkorbanalyse um gezielt Werbemaßnahmen einzuleiten.
  Bei 60% der Einkäufe, in denen Bier gekauft wird, werden auch Kartoffel-Chips gekauft. Beide Produkte kommen in 2% der Einkäufe vor.
- Viele Daten sind zu analysieren (Scannerkassen im Supermarkt, log-files im Internet, Rabattkarten, etc.). Merkmalskombinationen kommen spärlich vor.

Kenngrößen von Assoziationsregeln sind

- **Support**: relative Häufigkeit der Beispiele, in denen die Regel anwendbar ist. *.... kommen in 2% der Einkäufe vor.*
- **Konfidenz**: relative Häufigkeit der Beispiele, in denen die Regel richtig ist. *Bei 60% der Einkäufe, ...*

Algorithmen sind so zu entwerfen, dass alle gefundenen Assoziationsregeln a priori definierte **Mindestkonfidenz** und **Mindestsupport** erfüllen sollen.

Diese Verfahren sollen dabei keine Annahmen über die zu analysierenden Merkmale benötigen (wäre z.B. in einem Versandhandel mit vielen Tausend verschiedenen Artikeln auch nicht durchführbar).

(Schwenker 2004)

Beispiel:
Datenmatrix ("Transaktionen")

outlook	temperature	humidity	windy	play
sunny	hot	high	false	no
overcast	cool	normal	true	yes
sunny	cool	normal	false	yes
rainy	mild	high	true	no

Ein **Item** ist ein (Merkmal,Ausprägung)-Paar: $(\texttt{temperature}, \texttt{cool})$, $(\texttt{windy}, \texttt{true})$).

In typischen Anwendungen (wie Warenkorb-Analysen) mit binären Merkmalsausprägungen (aber mit vielen Merkmalen), sind nur einige wenige Merkmalsausprägung $= 1$.

Transaktionen sind effizienter als Mengen von Items speicherbar: z.B. $x_k = \{(\texttt{Bier}, 1), (\texttt{Chips}, 1)\}$ oder intuitiver $x_k = \{\texttt{Bier}, \texttt{Chips}\}$.

Assoziationsregeln

Eine **Assoziationsregel** $Y \to Z$ besteht aus einem

- **Regelrumpf** Y
- **Regelkopf** Z

wobei Y und Z zwei disjunkte Item-Mengen sind.

Ein Beispiel/Transaktion x_k aus der Datenmatrix/Transaktionenmenge erfüllt die Assoziationsregel $Y \to Z$, gdw. $Y \cup Z \subset x_k$.

Beispiel: Die Regel $(\texttt{temperature}, \texttt{cool}) \to (\texttt{humidity}, \texttt{normal})$ wird von den Beispielen/Transaktionen x_2 und x_3 erfüllt.

Für eine Item-Menge Y ist der *Support* definiert als

$$\text{support}(Y) := \frac{|\{x_k \in X \ : \ Y \subset x_k\}|}{n}$$

(n Anzahl der Beispiele in der Datenmatrix X).

Der *Support* für eine Assoziationsregel $Y \to Z$ zweier disjunke Item-Mengen Y und Z ist definiert durch:

$$\text{support}(Y \to Z) := \text{support}(Y \cup Z)$$

Die *Konfidenz* für eine Assoziationsregel $Y \to Z$ zweier disjunke Item-Mengen Y und Z ist definiert durch:

$$\text{konfidenz}(Y \to Z) := \frac{\text{support}(Y \to Z)}{\text{support}(Y)}$$

für "Support" auch: **Abdeckung**
für "Konfidenz" auch: **Genauigkeit**

Die Genauigkeit allein sagt wenig über eine Regel aus.

Gibt es beispielsweise einen extrem selten verkauften Artikel A im Warensortiment, der im betrachteten Zeitraum nur einmal verkauft wurde – zufällig zusammen mit den Artikeln A_1 und A_2 – hätte die Regel $A \Rightarrow A_1 \wedge A_2$ Genauigkeit 100%, da diese Regel sowieso nur durch ein einzelnes Itemset abgedeckt wird.

Für das Warenhaus ist diese Regel aber völlig wertlos.

Gesucht werden daher Regeln, die eine genügend große Abdeckung und genügend große Genauigkeit aufweisen.

Was als hinreichend häufig und hinreichend genau bezeichnet wird, hängt von der Anwendung ab.

Wenn beispielsweise eine Regel mit einer (relativen) Abdeckung von 30% und einer Genauigkeit von 10% sagt, dass Kunden, die eine persönliche Einladung zu einer Firmenpräsentation eines Autoherstellers mit Buffet erhalten, hinterher einen Wagen mit Luxusausstattung dieser Firma kaufen, ist dies eine sehr interessante Regel.

(Klawonn 2004)

Umgang mit numerischen Attributen: Quantisierung

Beispieldatensatz

outlook	temparature	humidity	windy	play
sunny	85	85	false	no
sunny	80	90	true	no
overcast	83	86	false	yes
rainy	70	96	false	yes
rainy	68	80	false	yes
rainy	65	70	true	no
overcast	64	65	true	yes
sunny	72	95	false	no
sunny	69	70	false	yes
rainy	75	80	false	yes
sunny	75	70	true	yes
overcast	72	90	true	yes
overcast	81	75	false	yes
rainy	71	91	true	no

nach Quantisierung

outlook	temparature	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

Beispiele für Assoziationsregeln aus dem rechten Datensatz:

$$\text{support}(\{(windy, true)\}) = 6/14$$

$$\text{support}(\{(temperature, cool), (humidity, normal)\}) = 4/14$$

$$\text{support}(\{(temperature, hot), (windy, false), (play, yes)\}) = 2/14$$

$$\text{support}(\{(temperature, cool)\} \rightarrow \{(humidity, normal)\}) = 4/14$$

$$\text{konfidenz}(\{(temperature, cool)\} \rightarrow \{(humidity, normal)\}) = 1$$

$$\text{konfidenz}(\{\{(humidity, normal)\} \rightarrow \{(temperature, cool)\}) = 4/7$$

$$\text{support}(\{(humidity, normal), (windy, false)\} \rightarrow \{(play, yes)\}) = 4/14$$

$$\text{konfidenz}(\{(humidity, normal), (windy, false)\} \rightarrow \{(play, yes)\}) = 1$$

Finden von Assoziationsregeln

Es seien gegeben

- Datenmatrix/Transaktionsmenge. Transaktionen x_k als Menge von Items.
- $s_{\text{min}} \in (0, 1)$ Wert für den minimalen Support.
- $k_{\text{min}} \in (0, 1)$ Wert für die minimale Konfidenz.

Gesucht sind alle Assoziationsregeln $Y \rightarrow Z$ mit

- support$(Y \rightarrow Z) \geq s_{\text{min}}$
- konfidenz$(Y \rightarrow Z) \geq k_{\text{min}}$

Apriori-Algorithmus: Grundidee

Zerlegung des Gesamtproblems in die beiden folgenden Teilprobleme:

1. Bestimme **alle** Item-Mengen X für die gilt:

   support$(X) \geq s_{\text{min}}$

   Item-Mengen X mit support$(X) \geq s_{\text{min}}$ sind die **häufigen** Item-Mengen.

2. Berechne **alle** Assoziationsregeln $X \rightarrow Y$ aus den häufigen Item-Mengen für die gilt:

   konfidenz$(X \rightarrow Y) \geq k_{\text{min}}$

Grundlagen

Monotonie-Eigenschaft

Jede Teilmenge eines häufig auftretenden Itemsets ist selbst auch häufig.

Vorgehen

- zuerst die einelementigen Frequent Itemsets bestimmen, dann die zweielementigen und so weiter
- Finden von k+1-elementigen Frequent Itemsets:

  nur solche k+1-elementigen Itemsets betrachten, für die alle k-elementigen Teilmengen häufig auftreten
- Bestimmung des Supports durch Zählen auf der Datenbank (ein Scan)

(Böhm 2003)

Bestimmung der häufigen Item-Mengen

Bei der Suche nach häufigen Item-Mengen macht man sich die folgende Eigenschaft von Item-Mengen zu Nutze:

Für zwei Item-Mengen mit $Y \subset Z$ gilt offenbar

$$\text{support}(Z) \leq \text{support}(Y)$$

Das heißt

- Teilmengen einer häufigen Item-Menge sind häufige Item-Menge.
- Obermengen einer nicht häufigen Item-Menge sind nicht häufige Item-Mengen.

Die häufigen Item-Mengen lassen sich iterativ (bzgl. ihrer Länge) bestimmen.

Definiere dazu für $l \geq 1$:

$$I_l := \{Y \ : \ \text{support}(Y) \geq s_{\text{min}}, |Y| = l\}$$

Die Berechnung von I_{l+1} aus I_l erfolgt in zwei Schritten:

1. Die l-elmentigen häufigen Item-Mengen aus I_l werden systematisch zu $l+1$ elementigen Item-Mengen erweitert.
   Für eine binärer Datenmatrix/Transaktionen gehen aus einer einzelnen l-elmentigen Item-Menge $n-l$ Item-Mengen der Kardinalität $l+1$ hervor.
   Dies sind zuznächst Kanditaten für häufige $l+1$ elementige Item-Mengen.

2. Jede dieser $l+1$ elementigen Item-Mengen Y muss anschließend geprüft werden, ob sie häufig ist, also ob $\text{support}(Y) \geq s_{\text{min}}$ gilt

A priori - Algorithmus:

1. Wähle $s_{\text{min}} \in (0,1)$; Setze $n = 1$ und $I = \emptyset$ und $H_1 = \{Y \ : \ Y$ ist 1-elementige Item-Menge$\}$
2. Bestimme nun den support(Y) für alle $Y \in H_n$ (ein Lauf durch die Transaktionsmenge!)
3. $I_n = \{Y \in H_n \ : \ \text{support}(Y) \geq s_{\text{min}}\}$
4. Falls $I_n = \emptyset$ return I sonst $I = I \cup I_n$
5. $H_{n+1} = \{Y \cup Y' \ : \ Y \in I_n$ und $Y' \not\subset Y$ mit $|Y'| = 1\}$
6. $n = n + 1$; Goto 2.

formalere Darstellung:

C_k: die zu zählenden Kandidaten-Itemsets der Länge k
L_k: Menge aller häufig vorkommenden Itemsets der Länge k

```
Apriori(I, D, minsup)
L1 := {frequent 1-Itemsets aus I};
k := 2;
while Lk-1 ≠ ∅ do
   Ck := AprioriKandidatenGenerierung(Lk-1);
   for each Transaktion T ∈ D do
      CT := Subset(Ck, T); // alle Kandidaten aus Ck, die in
        der Transaktion T enthalten sind;
      for each Kandidat c ∈ CT do c.count++;
   Lk := {c ∈ Ck | (c.count / |D|) ≥ minsup};
   k++;
return ∪k Lk;
```

Eigentliche Bildung der Assoziationsregeln:

Aus einer n-elementigen Item-Menge lassen sich 2^n-2 verschiedene Assoziationsregeln bilden, die allerdings möglicherweise nicht alle die vorgegebene Mindestkonfidenz haben.

Beispiel: Aus der 3-elementigen Item-Menge

$$X = \{(temperature, cool), (humidity, normal), (play, yes)\}$$

lassen sich die folgenden Assoziationsregeln bilden:

1. $(temperature, cool), (humidity, normal)\} \rightarrow \{(play, yes)$
2. $(temperature, cool), (play, yes)\} \rightarrow \{(humidity, normal)$
3. $(play, yes), (humidity, normal)\} \rightarrow \{(temperature, cool)$
4. $(temperature, cool)\} \rightarrow \{(humidity, normal), (play, yes)$
5. $(humidity, normal)\} \rightarrow \{(temperature, cool), (play, yes)$
6. $(play, yes)\} \rightarrow \{(temperature, cool), (humidity, normal)$

- Aus den häufigen Item-Mengen werden nun die Assoziationsregeln mit einer Konfidenz $\geq k_{\text{min}}$ erzeugt.
- Für zwei Item-Mengen X und Y mit $Y \subset X$ gilt offenbar, falls

  $$konfidenz((X \setminus Y) \rightarrow Y) \geq k_{\text{min}}$$

  so gilt für alle $Y' \subset Y$ ebenfalls

  $$konfidenz((X \setminus Y') \rightarrow Y') \geq k_{\text{min}}$$

- Zur Regelgenerierung nutzt man die Umkehrung. D.h. man beginnt mit einer möglichst kleinen Item-Menge Y' und schließt dann alle Item-Mengen Y mit $Y' \subset Y$ aus, falls schon gilt:

  $$konfidenz((X \setminus Y') \rightarrow Y') < k_{\text{min}}$$

- Man erzeugt aus einer häufigen Item-Menge X alle Regeln mit einer 1-elementigen rechten Seite. Also Regeln der Form $(X \setminus Y) \rightarrow Y$ mit $|Y| = 1$.
- Prüfe von diesen Regeln ob die Konfidenz $\geq k_{\text{min}}$. Diese Regeln werden ausgegeben.
- Sei H_l die Menge der Rechten Seiten von häufigen Item-Mengen mit l Elementen. Erzeuge aus H_l nun $l + 1$ elementige Item-Mengen H_{l+1}.
- Für alle Rechten Seiten $h \in H_{l+1}$ prüfe

  $$konfidenz((X \setminus h) \rightarrow h) \geq k_{\text{min}}$$

  Falls ja, dann gib die Regel aus, sonst h aus H_{l+1} entfernen.

(Schwenker '04)

Methoden der Effizienzverbesserung

Partitionierung der Datenbank [Savasere, Omiecinski & Navathe 1995]

- ein Itemset ist nur dann häufig, wenn es in mindestens einer Partition häufig ist
- bilde hauptspeicherresidente Partitionen der Datenbank

⇨ viel effizienter auf Partitionen, aber aufwendige Kombination der Teilergebnisse

Sampling [Toivonen 1996]

- Anwendung des gesamten Algorithmus auf ein Sample
- Zählen der gefundenen häufigen Itemsets auf der gesamten Datenbank
- Festellen evtl. weiterer Kandidaten und Zählen auf der gesamten Datenbank

Methode

- häufig vorkommender Itemset X
- für jede Teilmenge A von X die Regel $A \Rightarrow (X - A)$ bilden
- Regeln streichen, die nicht die minimale Konfidenz haben
- Berechnung der Konfidenz einer Regel $A \Rightarrow (X - A)$

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

- Speicherung der Frequent Itemsets mit ihrem Support in einer Hashtabelle

⇨ keine Datenbankzugriffe

Frage der "Interessantheit" von Assoziationsregeln

Motivation

Aufgabenstellung

• Daten über das Verhalten von Schülern in einer Schule mit 5000 Schülern

Beispiel

• Itemsets mit Support:

60% der Schüler spielen Fußball, 75% der Schüler essen Schokoriegel

40% der Schüler spielen Fußball *und* essen Schokoriegel

• Assoziationsregeln:

„Spielt Fußball" ⇒ „Isst Schokoriegel", Konfidenz = 67%

TRUE ⇒ „Isst Schokoriegel", Konfidenz = 75%

Fußball spielen und Schokoriegel essen sind *negativ korreliert*

Aufgabenstellung

• Herausfiltern von irreführenden Assoziationsregeln

• Bedingung für eine Regel $A \Rightarrow B$

$$\frac{P(A \cap B)}{P(A)} > P(B) - d$$

für eine geeignete Konstante $d > 0$

• Maß für die „Interessantheit" einer Regel

$$\frac{P(A \cap B)}{P(A)} - P(B)$$

• Je größer der Wert für eine Regel ist, desto interessanter ist der durch die Regel ausgedrückte Zusammenhang zwischen A und B.

Constraints für Assoziationsregeln

Motivation

- zu viele Frequent Item Sets

  Effizienzproblem

- zu viele Assoziationsregeln

  Evaluationsproblem

- manchmal Constraints apriori bekannt

  „nur Assoziationsregeln mit Produkt A aber ohne Produkt B“

  „nur Assoziationsregeln mit Gesamtpreis > 100 der enthaltenen Produkte“

➡ Constraints an die Frequent Itemsets

Typen von Constraints

[Ng, Lakshmanan, Han & Pang 1998]

Domain Constraint

- $S \theta v$, $\theta \in \{=, \neq, <, \leq, >, \geq\}$, z.B. $S.Preis < 100$
- $v \theta S$, $\theta \in \{\in, \notin\}$, z.B. $Snacks \notin S.Typ$
- $V \theta S$ oder $S \theta V$, $\theta \in \{\subseteq, \subset, \not\subset, =, \neq\}$, z.B. $\{Snacks, Weine\} \subseteq S.Typ$

Aggregations Constraint

$agg(S)\ \theta\ v$ mit

- $agg \in \{min, max, sum, count, avg\}$
- $\theta \in \{=, \neq, <, \leq, >, \geq\}$

z.B. $count(S.Typ) = 1$,
$avg(S.Preis) > 100$

Anwendung der Constraints

Bei der Bestimmung der Assoziationsregeln

- löst das Evaluationsproblem
- nicht aber das Effizienzproblem

Bei der Bestimmung der häufig auftretenden Itemsets

- löst evtl. auch das Effizienzproblem
- Frage bei der Kandidatengenerierung:

welche Itemsets kann man mit Hilfe der Constraints ausschließen?

Anti-Monotonie

Definition

Wenn eine Menge S ein anti-monotones Constraint C verletzt, dann verletzt auch jede Obermenge von S dieses Constraint .

Beispiele

- $sum(S.Preis) \leq v$ ist anti-monoton
- $sum(S.Preis) \geq v$ ist *nicht* anti-monoton
- $sum(S.Preis) = v$ ist *teilweise* anti-monoton

Anwendung

⇒ baue anti-monotone Constraints in die Kandidatengenerierung ein

(Böhm 2003)

Probleme des Apriori-Algorithmus:

- Im schlimmsten Fall ist Apriori exponentiell in R, weil womöglich alle Teilmengen gebildet würden.
  In der Praxis sind die Transaktionen aber spärlich besetzt.
  Die Beschneidung durch smin und confmin reicht bei der Warenkorbanalyse meist aus.
- Apriori liefert unglaublich viele Regeln.
- Die Regeln sind höchst redundant.
- Die Regeln sind irreführend, weil die Kriterien die apriori Wahrscheinlichkeit nicht berücksichtigen.
  Wenn sowieso alle Cornflakes essen, dann essen auch hinreichend viele Fußballer Cornflakes.

Aktuelle Forschungsgegenstände:

- Kondensierte Repräsentationen
- Bessere Kriterien als support und Konfidenz
- Anfrageoptimierung im Sinne induktiver Datenbanken durch constraints

- Hier sehen wir nur die ersten beiden Verbesserungen.
- Die Konferenzen KDD, PKDD und ICDM sind aber voll von Beiträgen zu „frequent itemsets".

"Kondensierte Repräsentationen":

Ersetzen der Datenbank bzw. der Baumstruktur durch eine kondensierte Repräsentation,

- die kleiner ist als die ursprüngliche Repräsentation und
- aus der wir alle häufigen Mengen und ihre Häufigkeit ableiten können, ohne noch mal die Daten selbst anzusehen.

Kondensierte Repräsentationen für Assoziationsregeln:

- Closed item sets
- Free sets

Operator, der die Menge aller Assoziationsregeln ableitet:

- Cover operator

In anderen Worten:

Wir hätten gern einen Versionenraum!
Der Versionenraum ist kleiner als der Hypothesenraum.
Außerhalb des Versionenraums kann das Lernziel nicht liegen.

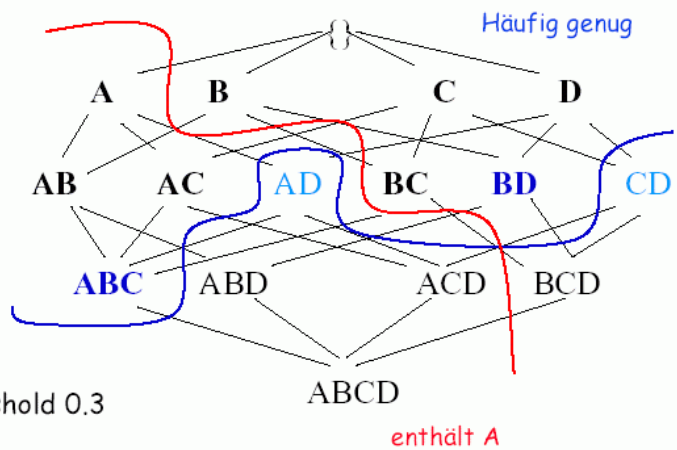
Wir müssen also aus den Beispielen

- eine untere Grenze und
- eine obere Genze konstruieren.

Eine Halbordnung bzgl. Teilmengenbeziehung haben wir schon.

Beispiel

A	B	C	D
1	0	1	0
1	1	1	0
0	1	1	1
0	1	0	1
1	1	1	0



Frequency threshold 0.3

Closed Item Sets

A	B	C	D
1	1	1	1
0	1	1	0
1	0	1	0
1	0	1	0
1	1	1	1
1	1	1	0

- closure(S) ist die maximale Obermenge (gemäß der Teilmengenbeziehung) von S, die noch genauso häufig wie S vorkommt.
- S ist ein *closed item set*, wenn closure(S)=S.
- Bei einem Schwellwert von 0,2 sind alle Transaktionen häufig genug.
- Closed sind: C, AC, BC, ABC, ABCD
  keine Obermenge von C kommt auch 6 mal vor;
  A kommt 5 mal vor, aber auch die Obermenge AC und keine Obermenge von AC
  ...

Kondensierte Repräsentation und Ableitung

Closed item sets sind eine kondensierte Repräsentation:

- Sie sind kompakt.
- Wenn man die häufigen closed item sets C berechnet hat, braucht man nicht mehr auf die Daten zuzugreifen und kann doch alle häufigen Mengen berechnen.

Ableitung:

- Für jede Menge S prüfen wir anhand von C:
  Ist S in einem Element X von C enthalten?
  - Nein, dann ist S nicht häufig.
  - Ja, dann ist die Häufigkeit von S ungefähr die von X.
    Wenn es in mehreren Elementen von C vorkommt, nimm die maximale Häufigkeit!

Freie Mengen (free sets)

- Eine Menge S ist frei, wenn es keine logische Regel (Konfidenz=1) zwischen ihren Elementen gibt, d.h.

$$\neg\exists X, Y \left| S = X \cup Y, Y \neq \{\ \}, X \Rightarrow Y \right.$$

- Eine Menge S ist δ-frei, wenn es keine Regel mit weniger als δ Ausnahmen zwischen ihren Elementen gibt.

- Die closed sets sind die closure der freien Mengen!
  Man kann die closed sets aus den freien Mengen berechnen.
- Freiheit ist eine anti-monotone Eigenschaft von Mengen.
  Deshalb kann man die freien Mengen effizient berechnen.

Beispiel

A	B	C	D
1	1	1	1
0	1	1	0
1	0	1	0
1	0	1	0
1	1	1	1
1	1	1	0

- Bei einem Schwellwert von 0,2 sind die häufigen freien Mengen:
  {}, A,B,D,AB

- Closed sind: C, AC, BC, ABC, ABCD

- Closure({})=C
  closure(A)=AC
  closure(B)= BC
  closure(D)=ABCD
  closure(AB)=ABC

Def.:

- Free(r, δ): Eine Menge X ist δ-frei, wenn es keine Regel zwischen ihren Elementen mit weniger als δ Ausnahmen gibt.
- Freq(r, σ): $\{X \mid X \subseteq R, |X \in r| / |r| \geq \sigma\}$
- FreqFree(r, σ, δ): Freq (r, σ) ∩ Free(r, δ)

MinEx

- Statt alle häufigen Mengen zu suchen, brauchen wir nur noch alle FreqFree(r, σ, δ) zu suchen.
- Bottom-up Suche im Halbverband der Mengen
  beginnt beim leeren Element, nimmt dann alle 1-elementigen Mengen,...
  endet bei den größten Mengen, die noch FreqFree(r, σ, δ) sind.
- Der Test, ob Mengen frei sind, erfordert das Bilden von strengen Regeln und erlaubt das Pruning der Mengen, in denen solche gefunden wurden.

Algorithmus von Jean-Francois Boulicaut

Eigenschaften von MinEx

- Der Algoritmus ist immer noch aufwändig, aber schneller als APRIORI und schneller als die Verwendung von closed sets.
- Der Algorithmus ist exponentiell in der Menge .
- Der Algorithmus ist linear in der Menge der Datenbanktupel, wenn δ im selben Maße steigt wie die Zahl der Tupel.
  Wir verdoppeln δ, wenn wir die Tupelzahl verdoppeln.
- Der Algorithmus approximiert das „wahre" Ergebnis.
  In der Praxis ist eine Abweichung von 0,3% aber kein Problem.

Zusammenfassung zu kondensierten Repräsentationen:

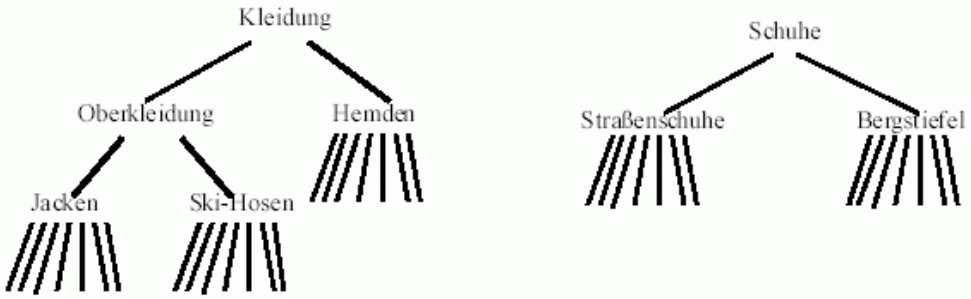
- Es gibt zwei Repräsentationen, die weniger Elemente für eine Suche nach häufigen Mengen ausgeben als eben alle häufigen Mengen. Aus diesen Repräsentationen können alle häufigen Mengen hergeleitet werden.
  - Die closed sets sind maximale Obermengen von S mit derselben Häufigkeit wie S.
  - Die free sets sind Mengen, aus denen man keine Assoziationsregeln machen kann.
- Wenn man die häufigen freien Mengen berechnet, hat man die untere Grenze im Versionenraum für Assoziationsregeln gefunden.
- Der Algorithmus MinEx findet diese Grenze.

(Morik 2003)

Hierarchische Assoziationsregeln

Motivation

• in vielen Anwendungen: Item-Taxonomien (*is-a* Hierarchien)



• suche auch Assoziationsregeln zwischen abstrakteren Items

z.B. zwischen Warengruppen

⇨ wesentlich höherer Support

Motivation

Beispiel

Ski-Hosen ⇒ Bergstiefel
Jacken ⇒ Bergstiefel
} Support < minsup

Oberkleidung ⇒ Bergstiefel Support > minsup

Eigenschaften

- Support von „Oberkleidung ⇒ Bergstiefel“ nicht unbedingt gleich
  Support von „Jacken ⇒ Bergstiefel“ + Support von „Ski-Hosen ⇒ Bergstiefel“
- wenn „Oberkleidung ⇒ Bergstiefel“ minimalen Support besitzt,
  dann auch „Kleidung ⇒ Bergstiefel“

Grundbegriffe [Srikant & Agrawal 1995]

- $I = \{i_1, ..., i_m\}$ eine Menge von Literalen, genannt „*Items*“
- H ein gerichteter azyklischer Graph über der Menge von Literalen I
- Kante in H von i nach j :
  i ist eine Verallgemeinerung von j,
  i heißt *Vater* oder *direkter Vorgänger* von j,
  j heißt *Sohn* oder *direkter Nachfolger* von i.
- $\overline{x}$ heißt *Vorfahre* von x (x *Nachfahre* von $\overline{x}$) bezüglich H:
  es gibt einen Pfad von $\overline{x}$ nach x in H
- Menge von Items $\overline{Z}$ heißt *Vorfahre* einer Menge von Items Z:
  mindestens ein Item in Z wird durch einen Vorfahren ersetzt

- D eine Menge von Transaktionen T, wobei $T \subseteq I$
- typischerweise:
  die Transaktionen T enthalten nur Items aus den Blättern des Graphen H
- Transaktion T *unterstützt ein Item* $i \in I$:
  i in T enthalten ist oder i ein Vorfahre eines Items, das in T enthalten ist
- T *unterstützt eine Menge* $X \subseteq I$ von Items:
  T unterstützt jedes Item in X
- *Support einer Menge* $X \subseteq I$ von Items *in* D :
  Prozentsatz der Transaktionen in D, die X unterstützen

- *hierarchische Assoziationsregel*:

  $X \Rightarrow Y$ mit $X \subseteq I$, $Y \subseteq I$, $X \cap Y = \emptyset$

  kein Item in Y ist Vorfahre eines Items in X bezüglich H

- *Support s einer hierarchischen Assoziationsregel* $X \Rightarrow Y$ *in* D :

  Support der Menge $X \cup Y$ in D

- *Konfidenz c einer hierarchischen Assoziationsregel* $X \Rightarrow Y$ *in* D:

  Prozentsatz der Transaktionen, die auch die Menge Y unterstützen in der Teilmenge aller Transaktionen, welche die Menge X unterstützen

Beispiel

TransaktionsID	Items
1	Hemd
2	Jacke, Bergstiefel
3	Ski-Hose, Bergstiefel
4	Straßenschuhe
5	Straßenschuhe
6	Jacke

Support von {Kleidung}: 4 von 6 = 67%
Support von {Kleidung, Bergstiefel}: 2 von 6 = 33%

„Schuhe ⇒ Kleidung“: Support 33%, Konfidenz 50%

„Bergstiefel ⇒ Kleidung“: Support 33%, Konfidenz 100%

Bestimmung der "frequent itemsets" im hierarchischen Fall:

Grundidee

- Erweiterung der Transaktionen der Datenbank um alle Vorfahren von enthaltenen Items
- Methode
  - jedes Item in einer Transaktion T wird zusammen mit all seinen Vorfahren bezüglich H in eine neue Transaktion T' eingefügt
  - es werden keine Duplikate eingefügt
- Bleibt zu tun:

  Finden von Frequent Itemsets für einfache Assoziationsregeln (Apriori-Algorithmus)

➡ *Basisalgorithmus* für hierarchische Assoziationsregeln

Optimierungen des Basisalgorithmus

Vorberechnung von Vorfahren

- zusätzliche Datenstruktur H

  Item $\longrightarrow$ Liste aller seiner Vorfahren

- effizienterer Zugriff auf alle Vorfahren eines Items

Filtern der hinzuzufügenden Vorfahren

- nur diejenigen Vorfahren zu einer Transaktion hinzufügen, die in einem Element der Kandidatenmenge C_k des aktuellen Durchlaufs auftreten
- Beispiel: C_k = {{Kleidung, Schuhe}}

  „JackeXY" durch „Kleidung" ersetzen

Ausschließen redundanter Itemsets

- Sei X ein k-Itemset, i ein Item und $\bar{i}$ ein Vorfahre von i.
- $X = \{i, \bar{i}, \ldots\}$
- Support von $X - \{\bar{i}\}$ = Support von X
- X kann bei der Kandidatengenerierung ausgeschlossen werden.
- Man braucht kein k-Itemset zu zählen, das sowohl ein Item i als auch einen Vorfahren $\bar{i}$ von i enthält. Redundanz: Bereits als $(k-1)$-intemset berücks.

➡ Algorithmus *Cumulate*

Stratifikation

- Alternative zum Basis-Algorithmus (Apriori-Algorithmus)
- Stratifikation = Schichtenbildung der Mengen von Itemsets
- Grundlage

  Itemset $\bar{X}$ hat keinen minimalen Support und $\bar{X}$ ist Vorfahre von X:
  X hat keinen minimalen Support.

- Methode
  - nicht mehr alle Itemsets einer bestimmten Länge k auf einmal zählen
  - sondern erst die allgemeineren Itemsets zählen

    und die spezielleren Itemsets nur zählen, wenn nötig

Beispiel

C_k = {{Kleidung Schuhe}, {Oberkleidung Schuhe}, {Jacken Schuhe} }

zuerst den Support für {Kleidung Schuhe} bestimmen

nur dann den Support für {Oberkleidung Schuhe} bestimmen, wenn {Kleidung Schuhe} minimalen Support hat

Begriffe

- *Tiefe* eines Itemsets:

  Für Itemsets X aus einer Kandidatenmenge C_k ohne direkten Vorfahren in C_k: $Tiefe(X) = 0$.
  Für alle anderen Itemsets X in C_k:
  $Tiefe(X) = \max\{Tiefe(\overline{X}) \mid \overline{X} \in C_k$ ist direkter Vorfahre von $X\} + 1$.

- (C_k^n): Menge der Itemsets der Tiefe n aus C_k, $0 \leq n \leq$ maximale Tiefe t

Algorithmus *Stratify*

- Zählen der Itemsets aus C_k^0
- Löschung aller Nachfahren von Elementen aus (C_k^0), die keinen minimalen Support haben
- Zählen der übriggebliebenen Elemente aus (C_k^1)
- und so weiter . . .

Tradeoff zwischen Anzahl der Itemsets, für die Support auf einmal gezählt wird und der Anzahl von Durchläufen durch die Datenbank

⇒ $|C_k^n|$ klein, dann Kandidaten der Tiefen (n, n+1, ..., t) auf einmal zählen

Problem von *Stratify*

falls sehr viele Itemsets mit kleiner Tiefe den minimalen Support haben:

Ausschluß nur weniger Itemsets größerer Tiefe

Verbesserungen von *Stratify*

- Schätzen des Supports aller Itemsets in C_k mit einer Stichprobe
- C_k': alle Itemsets, von denen man aufgrund der Stichprobe erwartet, daß sie oder zumindest alle ihre Vorfahren in C_k minimalen Support haben
- Bestimmung des tatsächlichen Supports der Itemsets in C_k' in einem Datenbankdurchlauf
- Entfernen aller Nachfahren von Elementen in C_k', die keinen minimalen Support haben, aus der Menge C_k'', $C_k'' = C_k - C_k'$
- Bestimmen des Supports der übriggebliebenen Itemsets in C_k'' in einem zweiten Datenbankdurchlauf

Experimentelle Untersuchung

Testdaten

- Supermarktdaten

  548000 Items, Item-Hierarchie mit 4 Ebenen, 1,5 Mio. Transaktionen

- Kaufhausdaten

  228000 Items, Item-Hierarchie mit 7 Ebenen, 570000 Transaktionen

Ergebnisse

- Optimierungen von *Cumulate* und Stratifikation können kombiniert werden
- die Optimierungen von *Cumulate* bringen eine starke Effizienzverbesserung
- die Stratifikation bringt nur noch einen kleinen zusätzlichen Vorteil

(Böhm 2003)

bisher:
Bewertung von Assoziationsregeln nach statistischer Sichtweise (es darf Ausnahmen geben, diese dürfen nicht zu häufig sein)

auch möglich:
alle *exakten* (auf der gesamten Datenmenge gültigen) Regeln extrahieren

dabei hilft der Begriffsverband

wir beschränken uns wieder auf einwertige Kontexte (d.h. binäre Attribute)

(G, M, I) sei einwertiger Kontext im Sinne der Begriffsanalyse (vgl. Kapitel 4)

Formale Definition von Implikationen

Def: Sei $A,B \subseteq M$, $T \subseteq M$ respektiert eine Implikation $A \rightarrow B$, wenn $A \not\subseteq T$ oder $B \subseteq T$ ist. $A \rightarrow B$ gilt in einem Kontext (G,M,I), wenn sie im System der Gegenstandsinhalte gilt. A heißt dann Prämisse für B.

Satz: Eine Implikation $A \rightarrow B$ gilt in (G,M,I) genau dann, wenn $B \subset A''$ ist.

Wie lassen sich die Implikationen am Begriffsverband ablesen?

- $A \rightarrow m$ gilt genau dann, wenn $(m',m'') \geq (A',A'')$ $(\Leftrightarrow \mu m \geq \wedge\{\mu n | n \in A\})$.
- Man muß also prüfen, ob im Diagramm der mit m bezeichnete Begriff über dem Infimum aller mit einem n aus A bezeichneten Begriffe liegt.

Die Menge der Implikationen

- Eine Implikation $A \rightarrow B$ folgt semantisch aus einer Menge $\mathcal{L}$ von Implikationen, falls jede Teilmenge von M, die $\mathcal{L}$ respektiert, auch $A \rightarrow B$ respektiert.
- $\mathcal{L}$ heißt abgeschlossen, wenn sie alle Implikationen des Kontextes (G,M,I) enthält.
- $\mathcal{L}$ heißt vollständig, wenn jede Implikation von (G,M,I) aus $\mathcal{L}$ folgt.

Welche Implikationen sind notwendig, um einen Kontext wiederzugeben?

- Einige Implikationen sind trivial oder ergeben sich offensichtlich aus anderen:
  - A→B gilt stets, wenn B⊆A
  - wenn A→B und C⊆B, dann A→C
  - aus $A_j \to B_j$ für $j \in J$, folgt $\bigcup_{j\in J} A_j \to \bigcup_{j\in J} B_j$
- eliminiert man diese Implikationen, bleiben die Implikationen mit echter Prämisse übrig, diese Menge ist vollständig.

A⊆M ist echte Prämisse, wenn $\varnothing \neq A^{\bullet} := A'' \setminus (A \cup \bigcup_{n\in A}(A \setminus \{n\})'')$

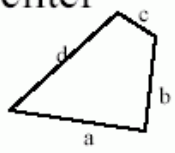
Stammbasis (Duquenne-Guigues-Basis)

- Eine Menge $\mathcal{L}$ von Implikationen heißt nichtredundant, wenn keine der Implikationen aus den anderen folgt.
- Eine nichtredundante vollständige Menge $\mathcal{L}$ bildet eine Basis.
- Die Menge $L := \{P \to (P'' \setminus P) \mid P \text{ Pseudoinhalt}\}$ bildet eine Basis (Stammbasis).

P⊆M heißt Pseudoinhalt, wenn P≠P'' und für jeden Pseudoinhalt Q⊆P, Q≠P bereits Q''⊆P gilt.

Noch einmal die Vierecke:

- Aus der Liste der Implikationen mit echter Prämisse ist die Menge aller gültigen Implikationen leichter abzulesen, als aus der Stammbasis.
  - Bsp: c=d, b=d ⇒ a=b, a=c, a||c, b||d.



Implikationen mit echter Prämisse

1 : c=d ==> a=b
2 : a=b ==> c=d
3 : b=d ==> a||c
4 : b||d ==> a||c a=c b=d
5 : a=c ==> a||c b||d b=d
6 : a senk b ==> a||c b||d a=c b=d
7 : c=d b=d ==> b||d a=c
8 : a=b b=d ==> b||d a=c
9 : a||c c=d ==> b||d a=c b=d
10 : a||c a=b ==> b||d a=c b=d

Stammbasis

1 : a senk b ==> a||c b||d a=c b=d
2 : b=d ==> a||c
3 : a=c ==> a||c b||d b=d
4 : c=d ==> a=b
5 : a=b ==> c=d
6 : b||d ==> a||c a=c b=d
7 : a||c a=b c=d ==> b||d a=c b=d

Erzeugung von Kontexten aus Implikationen

- Es ist möglich einen Kontext aus der Merkmallogik zu erschließen und umgekehrt.

⇒ Die Menge der Implikationen eines Kontextes enthält dieselbe Information wie der Kontext selbst.

- Manchmal wissen wir mehr über die Implikationen eines Kontextes als über seine Gegenstände/Merkmale. ConImp erlaubt es, den Kontext interaktiv aus den Implikationen zu erstellen (Merkmalexploration).

Interaktives Erstellen eines Auswahlkontextes

- gegeben:
  - umfangreiches, nicht strukturiertes Wissen über ein Sachgebiet;
  - Menge von Eigenschaften (Merkmalen);
  - Gegenstandsbereich (Universum).
- gesucht:

  minimale Teilmenge des Gegenstandsbereichs, für die genau die Merkmalsimplikationen des gesamten Universums gelten („trennende Beispiele“, „vollständiger Auswahlkontext“ ≅ Kontext).

Interaktives Erstellen eines Kontextes mit ConImp

Beispiel: Vierecke

- Gebe Merkmale an, die Vierecke charakterisieren

  $(a=b,\ a\|c,\ a\perp b,\ \ldots)$.
- ConImp fragt Implikationen ab.

  $(a\|c \wedge b\|d \rightarrow a=c \wedge b=d\ ?)$
- bestätige gültige Implikationen (beweisbar).
- ist die Implikation ungültig, konstruiere ein Gegenbeispiel.

  $(a\|c \wedge b=d \rightarrow a=c\ ?$ Nein, denn $\{(0,0),(0,4),(1,1),(3,1)\})$

(Ganter & Wille, o.J.)