Frequent Pattern Discovery

Marius Morawski

December 17, 2003

Contents

1 Einleitung

Ein wichtiges Problem beim Datamining ist, Regelmäßigkeiten und Muster in großen Datenbestnden zu finden.
Dieses Problem tritt zum Beispiel auf, wenn das Kaufverhalten in Supermärkten oder die Zugriffsstatistiken einer Website analysiert werden sollen.
Im Supermarkt- Beispiel wären solche Muster z.B. gewisse Produkte, die häufig zusammen gekauft werden. Z.B. könnte es sein, dass unter den Kunden, die Käse kaufen, auffällig viele auch Wein kaufen.
Solche Daten liegen dann in einer Form wie der folgenden vor:

Basket-ID	A	B	C	D	E
t_1	1	0	0	0	0
t_2	1	1	1	1	0
t_3	1	0	1	0	1
t_4	0	0	1	0	0
t_5	0	1	1	1	0
t_6	1	1	1	0	0
t_7	1	0	1	1	0
t_8	0	1	1	0	1
t_9	1	0	0	1	0
t_{10}	0	1	1	0	1

Diese Matrix ist so zu verstehen, dass jede Zeile eine Transaktion darstellt (im Beispiel also einen Einkauf eines Kunden) und die Einsen bzw. Nullen sagen aus, ob bei Transaktion t Item A,, B, C, ... vorkommt (im Beispiel: ob Produkt x gekauft wurde).
Nun kann man sich leicht vorstellen, dass solche Datenbestände gewaltige Ausmaße annehmen können – um beim Supermarkt- Beispiel zu bleiben: In der Regel gibt es eine gewaltige Anzahl von Produkten und auch die Anzahl der Transaktionen kann sehr gro
ss werden. Es ergibt sich also das Problem, wie man mit solchen Datenbeständen effizient umgeht und nutzbare Informationen daraus gewinnen kann – eben z.B., welche Produkte oft gemeinsam im Einkaufskorb landen.
Genau dieses Problem wird bei der Frequent Pattern Dicovery thematisiert.

2 Grundlegendes zur "Frequent Pattern Discovery"

Frequent Patterns sind, wie der Name schon sagt, "Häufige Muster", also Muster, die in einem Datenbestand "häufig" sind.
Wie schon in der Einleitung dargestellt, ist die Größe des Datenbestandes ein wichtiges Problem, das angegangen werden muss, um die Daten nutzen zu können. Die Strategie, "häufige Muster" zu finden, indem man einfach alle Kandidaten in allen möglichen Kombinationen aus beliebig vielen Items durchprobiert und ausrechnet, wie häufig die jeweilige Kombination ist, kommt daher offensichtlich nicht in Frage – das wäre nicht mit vertretbarem Aufwand durchzuführen.
Glücklicherweise haben die gesuchten Muster Eigenschaften, die es erlauben, die größte Anzahl der möglichen Muster nicht in Betracht ziehen zu müssen. Insbesondere können enthalten alle komplexeren Muster ausschließlich einfachere Muster, die selbst häufig sind. Wenn z.B. Käse schon nicht besonders häufig gekauft würde, kann auch die Kombination Käse + Wein nicht mehr häufig sein und muss garnicht erst in Betracht gezogen werden!
Abstrahierend kann man sagen, dass zwischen Mustern eine Generalisierungs-/Spezialisierungs- Beziehung besteht.
Um diese Generalisierungs- /Spezialisierungs- Beziehung bei der Suche nach Frequent Patterns zu nutzen, bietet sich ein Bootom-up-Ansatz an, bei dem zuerst einfachste Muster (die aus einzelnen Items bestehen) gesucht werden, dann in diesen Mustern nach größeren häufigen Mustern gesucht wird, und darin wiederum nach größeren.

Im folgenden werden diese Ideen formalisiert und konkretisiert. Des weiteren werden auch einige Abwandlungen und Varianten davon vorgestellt.

3 Assoziationsregeln

3.1 Regeln allgemein

Zur Beschreibung von Zusammenhängen hat sich der regelbasierte Ansatz als besonders nützlich herausgestellt. Besonders in der Forschung zu Künstlicher Intelligenz hat sich dieser Ansatz bewährt.
Allgemein sind Regeln nach einem einfachen Wenn-Dann-Muster aufgebaut: "Wenn A, dann B" – eine einfache Implikation also: $A \Rightarrow B$
Eine häufige Abwandlung sind probabilistische Regeln: Wenn A, dann mit Wahrscheinlichkeit P auch B: $A \Rightarrow B[P]$

Generell ist dieser Ansatz gut geeignet zur Repräsentation von Kategorisierungen (Mengen lassen sich ja dadurch definieren, welche Objekte ihnen angehöhren und welche nicht). Man kann das auf Variablen mit reellen Zahlen erweitern (z.B. "$X > 10,2 \Rightarrow Y > 5,2''$).
Ein Vorteil des regelbasierten Ansatzes ist auch, dass Regeln dem menschlichen Vertändnis entgegenkommen, da sie auf einfache Weise logische Zusammenhänge beschreiben (dieser Vorteil besteht allerdings nur, solange die Menge der Regeln übersichtlich bleibt). In der Regel lässt man auf der rechten und linken Seite der Implikation nur Konjunktionen zu, sodass sich Regeln von der Form "$A_1 \bigwedge A_2 \bigwedge A_3 \bigwedge \ldots \Rightarrow B_1 \bigwedge B_2 \bigwedge B_3 \bigwedge \ldots$" ergeben. Regeln dieser Form sind sehr viel einfacher und effizienter zu verarbeiten als Regeln beliebiger Form. Meistens lässt man außerdem auf der rechten Seite der Implikation nur Konjunktionen mit nur einer Klausel zu, was die Sache weiter vereinfacht.

Bei der Frequent Pattern Discovery kommen Assoziationsregeln zum Einsatz - im folgenden Abschnitt werden diese noch etwas formalisiert.

3.2 Definitionen zu Assoziationsregeln

Einige wichtige Definitionen zu Assoziationsregeln:

Itemset: Ein Muster θ von der Form $(A_{i_1} = 1) \bigwedge (A_{i_2} = 1) \bigwedge (A_{i_2} = 1) \bigwedge \ldots (A_{i_k} = 1)$ heißt Itemset (Item-Menge)

Assoziationsregel: Eine Regel von der Form $\theta \Rightarrow \gamma$ heißt Assoziationsregel, wenn θ und γ Itemsets sind und γ aus nur einer Klausel besteht

Häufigkeit $frq(\theta)$: Anzahl der Fälle im Datenbestand, die θ erfüllen (wenn man θ als Menge auffasst, Mächtigkeit der Menge)

Rückhalt: $frq(\theta \Rightarrow \gamma)$ nennt man den Rückhalt der Regel $\theta \Rightarrow \gamma$

Konfidenz: $c(\theta \Rightarrow \gamma) = \frac{frq(\theta \Rightarrow \gamma)}{frq(\theta)}$ nennt man Konfidenz der Regel $\theta \Rightarrow \gamma$
Man kann $c(\theta \Rightarrow \gamma)$ als Abschätzung der bedingten Wahrscheinlichkeit $P(\gamma|\theta)$ oder bezogen auf Regeln als P mit $\theta \Rightarrow \gamma[P]$ ansehen.

Border/Grenze $Bd(S)$: Der Border einer Menge S von Itemsets ist die Menge der Itemsets, deren sämtliche General-

isierungen zu S gehören, deren sämtliche Spezialisierungen aber nicht mehr zu S gehören.

der negative Border $Bd^-(S)$ alle Itemsets, die gerade nicht mehr zu S gehören

der positive Border $Bd^+(S)$ alle Itemsets, die gerade noch zu S gehören

$Bd(s) = Bd^-(S) \bigcup Bd^+(S)]$
(Anmerkung: Mit dem Border einer Menge von Frequent Itemsets wird die Menge der Frequent Itemsets komplett beschrieben - dafür reicht es sogar, nur $Bd^-(S)$ oder nur $Bd^-(S)$ anzugeben)

3.3 Finden von Assoziationsregeln

Jetzt, da die theoretischen Grundlagen geschaffen sind, kommen wir nun zum Findene von Assoziationsregeln.
Beim Suchen von Assoziationsregeln ist man an Regeln interessiert, die einen möglichst großen Rückhalt und möglichst hohe Konfidenz im gegebenen Datensatz haben – Regeln mit zu geringem Rückhalt machen nur Aussagen über einen sehr kleinen Anteil der Daten und sind damit nicht aussagekräftig.
Die Suche soll so effizient wie möglich gestaltet werden. Insbesondere sollten möglichst wenig Durchläufe auf dem Gesamtdatenbestandnotwendig sein, weil diese Daten in der Regel wegen ihrem Umfang nicht in den Arbeitsspeicher passen und der Zugriff damit entsprechend zeitintensiv ist. Man kann sich dabei zunutze machen, dass es im Datenbestand meist nur relativ wenige Frequent Itemsets gibt (um zum Supermarkt- Beispiel zurückzukehren: Es gibt eine große Anzahl von Produkten, aber jeder einzelne Kunde kauft nur einen kleinen Anteil der Waren aus dem Angebot). Aus diesen Frequent Itemsets lassen sich dann recht einfach Assoziationsregeln finden.

Man unterteilt die Suche nach Assoziationsregeln daher in zwei Schritte:

1. Suche aus einem Datenbestand alle Frequent Itemsets:
   - Lege Mindest- Häufigkeit s fest
   - Finde alle Itemsets, die mindestens s mal vorkommen
2. Suche in diesen Itemsets nach Assoziationsregeln, deren Konfidenz ausreichend ist:
   - Lege Mindest- Konfidenz c für Assoziationsregeln fest
   - Finde alle Assoziationsregeln, deren Konfidenz mindestens c ist

In der Beispiel- Matrix aus der Einleitung findet man z.B. für $s = 0,4$ die Itemsets $\{A\}, \{B\}, \{C\}, \{D\}, \{AC\}, \{BC\}$. Daraus könnte man z.B. die Regeln $A \Rightarrow C$ (Konfidenz 2/3) und $B \Rightarrow C$ (Konfidenz 1) finden (Anmerkung: die Häufigkeit des Itemsets, aus dem eine Regel hervorgegangen ist, ist dann nach Definition automatisch der Rückhalt der Regel).

3.3.1 Finden von Frequent Itemsets

Ein grundlegender Algorithmus zum Finden von Frequent Itemsets ist der APriori-Algorithmus. Er basiert auf der in Abschnitt 2 auf Seite 3 schon vorgestellten Idee Der Bottom-Up-Vorgehensweise und sucht einfache häufige Muster, die dann zur Suche komplexerer Muster weiterverwendet werden. Der Algorithmus läuft wie folgt ab (unoptimierter Pseudo-Code):

```
i=0;
C_i={{A}|A ist Variable};

While (C_i not empty)
{
  L_i = leere Menge von Sets der Groesse i

  //Suchen von L\"osungen
  For (each set S in C_i)
  {
    If (S.isFrequent()) then L_i.add(S);
  }

  //Erzeugung von Kandidaten f\"ur den n\"achsten Schritt
  C_i+1 = leere Menge von Sets der Groesse i+1;

  For (each set S in L_i)
  {
    For (each set T in L_i after S)
    {
      If (size(S U T)=i+1) then C_i+1.addEntry(S U T);
    }
  }

  i++;
}
```

Wie man sieht, bearbeitet der Algorithmus in jedem Schritt der While-Schleife eine Menge von Kandidaten `C_i` der Länge `i`. Im ersten Schritt ist dese Menge der Kandidaten die Menge aller Sets, die genau eine Variable enthalten, Für jedem folgenden Schritt wird aus der Menge der gefundenen Frequent Sets `L_i` der Länge `i` im vorhergehenden Schritt eine Menge von Kandidaten der Länge `i+1` erzeugt.
Die While-Schleife wird solange durchlaufen, bis keine geeigneten Kandidaten für den nächsten Schritt mehr gefunden werden.

Zur Laufzeit- Komplexität dieses Algorithmus:

Finden der Frequent Itemsets: Für das Finden der Lsungen ist nur ein Durchlauf durch den Datenbestand nötig, bei dem für jedes Itemset ein Zähler für die Häufigkeit mitläuft - $O(n)$ also.

Finden von Kandidaten für die nächste Runde Es gibt $(|\texttt{L\_i}|^2$ viele Kombinationen, die in Betracht gezogen werden müssen, das Kombinieren zu einem neuen Set benötigt jeweils eine Anzahl von Schritten in der Größenordnung $|\texttt{L\_i}|$ - daraus ergibt sich eine Worst-Case- Komplexität von $O(n^3)$. Wegen der Struktur der Daten (dünn besetzt) ergibt sich jedoch in der Regel eine Laufzeit- Komplexität von $O(n)$.

Mögliche Optimierungen für diesen Algorithmus können an verschiedenen Punkten ansetzen:

Reduzierung der Anzahl der Durchläufe durch Sampling
Bei häufigen Itemsets kann man in der Regel davon ausgehen, dass sie in einem Sample der Daten auch häufig sind. Deswegen kann man folgendermaßen vorgehen:

1. Den selben Algorithmus auf dem Sample laufen lassen (geforderten Mindestwert s leicht senken)
2. Häufigkeit der gefundenen Frequentz Sets im Gesamtdatenbestand ausrechnen
3. Falls es Sets gibt, die selbst nicht als häufig erkannt wurden, die aber ausschließlich Häufige Subsets haben, muss ein weiterer Durchlauf auf dem Gesamtdatenbestand gestartet werden, um sicherzustellen, dass keine Frequent Sets übersehen wurden

In der Regel braucht man damit nur zwei mal den Gesamtdatenbestand durchgehen.

Zahl der möglichen Kandidaten eingrenzen

Berechnung der Häufigkeit beschleunigen z.B. durch Speicherung der Daten als Baumstruktur

Entsprechend setzen Alternativen zu APriori auch an verschiedenen Punkten an:

DIC Hier wird bereits während der Datenbankzugriffs mit der Überprüfung von Itemsets auf äufigkeit begonnen

Partition Führt die suche nach Frequent Itemsets auf Teilen der Datenbank durch, die nacheinander in den Speicher geladen werden und prüft das Ergebnis anschließendmit dem Gesamtdatenbestand ab.

Algorithmus mit Hashing (Park) Kann viele der nicht-häufigen Itemsets ohne Datenbankzugriff ausschließen. Zusätzlich wird die datenbank in jedem Schritt verkleinert, um die Effizienz folgender Schritte zu steigern.

Randomisierter Algorithmus (Gunopulos) Gefundene Muster werden jeweils so lange versuchsweise erweitert, bis ein entstehende Muster nach den gesetzten Kriterien nicht mehr häufig genug ist. Die dazu nötigen randomisierten Einzel-Zugriffe auf die Datenbank sind aber der Effizienz nicht besonders zuträglich.

MaxEclat/ MaxClique Vor der Suche nach APriori wird bereits nach möglichst großen Frequent Itemsets gesucht – diese (inklusive ihrer deren Subitemsets) müssen dann bei der anschließenden APriori- Suche nicht mehr betrachtet werden (siehe dazu auch die Definition zu Borders in Abschnitt 3.2 auf Seite 5).

Picer-Search Ähnliche Idee wie bei MaxEclat/ MaxClique, jedoch wird während der gesamten Suche gezielt nach maximal grossen Itemsets gesucht.

MaxMiner Sehr ähnlich Pincer.

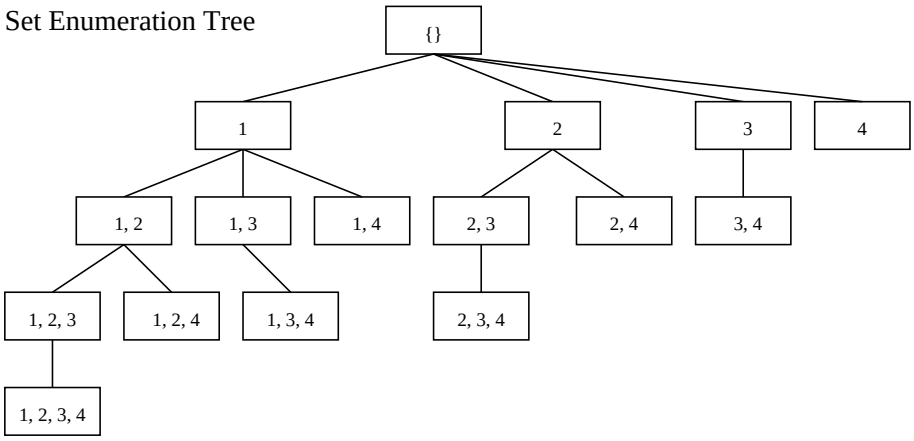
Im Folgenden wird der MaxMiner- Algorithmus exemplarisch als Alternative zu APriori noch etwas näher erläutert.
Die beiden Hauptideen dabei sind:

1. Während der gesamten Suche wird gezielt nach maximal häufigen (d.h. die größten, die gerade noch häufig genug sind) Itemsets gesucht - statt

alle Itemsets aufzuzählen, versucht man also, möglichst schnell bis zum (positiven) Border der Menge der Frequent Itemsets vorzustoßen und so viele Itemsets garnicht erst betrachten zu müssen (siehe dazu wiederum die Definition zu Borders in Abschnitt 3.2 auf Seite 5).

2. Zur Bestimmung, ob ein Itemset häufig ist, wird eine untere Grenze aus den Daten, die aus vorigen Datenbankzugriffen vorliegen, berechnet - falls diese untere Grenze über dem geforderten Mindestwert für s liegt, braucht man keinen Zugriff auf die Datenbank zur Bestimmung der Häufigkeit

Zur Umsetzung dieser Ideen benutzt der MaxMiner-Algorithmus eine Suchstrategie, die auf dem Prinzip eines Set-Enumeration-Tree basiert.



Set Enumeration Trees stellen Mengen dar, indem hauf oberster Ebene des Baums die kleinsten Teilmengen (aus einem einzelnen Element bestehend) "eigehängt" werden und dann sukzessive als Unterknoten davon die kleinsten echten Obermegen (also jeweils ein Element größer) des jeweiligen Knotzens.

Beim MaxMiner-Algorithmus wird dieser Set Enumeration Tree nicht explizit erzeugt, sondern in jedem Schritt werden nur jeweils einen Knoten (und dessen Unterknoten) behandelt. Gespeichert wird der jeweils aktuelle Knoten samt Unterknoten (kandidatengruppe g) als zwei Mengen:

Kopf ("head") $h(g)$ Diese Menge ist die durch den aktuell behandelten Knoten definierte Untermenge von Knoten (entspricht in der obigen Abbildung jeweils der Beschriftung der Knoten).

Rest("tail") $t(g)$ Enthält alle Elemente, die in Unterknoten, nicht aber im Knoten selbst enthalten sind.

In der obigen Abbildung z.B. ist zum Knoten "1,2" $1, 2$ der Kopf und $3, 4$ der Rest.
Mit diser Reprsentation können folgende Grundsätze umgesetzt werden:

Maximal häufige Itemsets suchen $fr(h(g) \bigcup (t(g)) > minfreq$: Wenn die in diesem Ast maximal audtretende Obermenge (nämlich die Vereinigung von h und t) häufig ist, sind wir auf den Border gestoßen und brauchen die restlichen Unterknoten nicht mehr explizit abprüfen, denn sie sind automatisch auch häufig.

nicht häufige Itemsets wegfallen lassen $fr(h(g) \bigcup \{i \in t(g)\}) > minfr$: Heißt, dass das Itemset (und damit auch Supersets/Unterknoten davon) nicht häufig sind und -wie schon bei APriori- wegfallen)

Im MaxMiner- Algorithmus sind außerdem noch folgende Optimierungsideen realisiert:

- Das eigentliche Ziel ist ja, möglichst schnell maximal häufige Itemsets (also zum positiven Border gehörige) zu finden - es sollte also möglichst oft $fr(h(g) \bigcup (t(g)) > minfreq$ gelten. Wie kann man das fördern? Indem man den Baum so umsortiert, dass häufige Elemente in möglichst vielen Ästen auftauchen! Das lässt siuch erreichen, indem man die Menge so umsortiert, dass häufige Items mglichst weit hinten auftreten - wie man an der Abbildung sieht, tritt die 4 wegen der dort gewählten Sortierung der Menge überall entweder im h oder in t auf.

- die Häufigkeit von Unterknoten durch bereits im Speicher vorliegende Informationen nach unten abschätzen - wenn diese Abschätzung positiv ausfällt, kann man den Unterknoten ohne aufwendigen Datenbankzugriff als häufig erkennen.

In diesem Abschnitt wurden Methoden und Algorithmen vorgestellt, um Frequent Itemsets zu finden. Im folgenden Abschnitt soll es nun darum gehen, aus gefundenen Frequent Itemsets aussagefähige Assoziationsregeln zu generieren.

3.3.2 Erzeugung von Assoziationsregeln

Das Finden von Assoziationsregeln aus gefundenen Frequent Itemsets erfolgt auf recht primitive Weise:
Aus jedem Itemset der Form $A_{i_1} \bigwedge A_{i_2} \bigwedge A_{i_2} \bigwedge \ldots A_{i_k}$ können k Regeln der Form $A_{i_1} \bigwedge A_{i_2} \bigwedge A_{i_2} \bigwedge \ldots A_{i_k-1} \Rightarrow A_{i_k}$ erzeugt werden. Hier ist man aber nur

an denjenigen Regeln interessiert, die auch mindestens in einem bestimmten Prozentsatz der Fälle zutreffen - kurz: Die Konfidenz der Regel muss einen Mindestwert erreichen.
Entsprechend läuft auch die Suche ab:

1. Konfidenzwert c festlegen
2. für alle möglichen Regeln den Konfidenzwert errechnen und nur die akzeptieren, die ihn erfüllen - dafür reicht ein Durchlauf durch den Gesamtdatenbestand aus.

Damit hat man dann eine Menge von Regeln, die tatsächlich für eine ausreichend große Teilmege der Daten zutreffend sind.

3.3.3 Relevanz von gefundenen Assoziatinsregeln

Nachdem wir jetzt also eine Menge von Assoziationsregeln gefunden haben, bleibt die Frage, ob die gefundenen Regeln tatsächlich relevant und aussagekräftig sind. Das ist leider nicht unbedingt immer der Fall:
In einer medizinischen Datenbank mag z.B. die Regel, dass ein Patient, der schwanger ist, mit sehr hoher Konfidenz weiblich ist, entdeckt werden, aber für den Benutzer ist es trivial.
Die Konfidenz (und damit auch die Häufigkeit) alleine ist also kein ausreichendes Kriterium für relevante Regeln. Welche Kriterien kann man sonst benutzen?
Eine Möglichkeit ist, statistische Signifikanz als Maßfür Interessantheit zu nutzen - falls also z.B. $P(B =!|A =!) \approx P(B)$, bedeutet das für eine Regel $A = 1 \Rightarrow B = 1$, dass sie uns nichts neues sagt. Manche trivialen Regeln wie das Beispiel mit der Schwangerschaft können aber auch damit nicht eliminiert werden - dazu benötigt man Anwendungswissen. Trotzdem ist die statistische Signifikanz in vielen Fällen eine geeignete Methode (wenn z.B. kein Anwendungswissen vorhanden ist oder als Ergänzung). Im Abschnitt 4.3 auf Seite 17 wird darauf noch näher eingegangen werden.

4 Erweiterung auf andere Problemstellungen

Die bisher vorgestellten Methoden sind durch ihre Definition in ihren Anwendungsgebieten eingeschränkt. Es ist jedoch problemlos möglich, andere ähnliche die vorgestellten Algorithmen so zu verallgemeinern, dass sie auch auf anderen Gebieten mit verwandten Problemstellungen zur anwendung

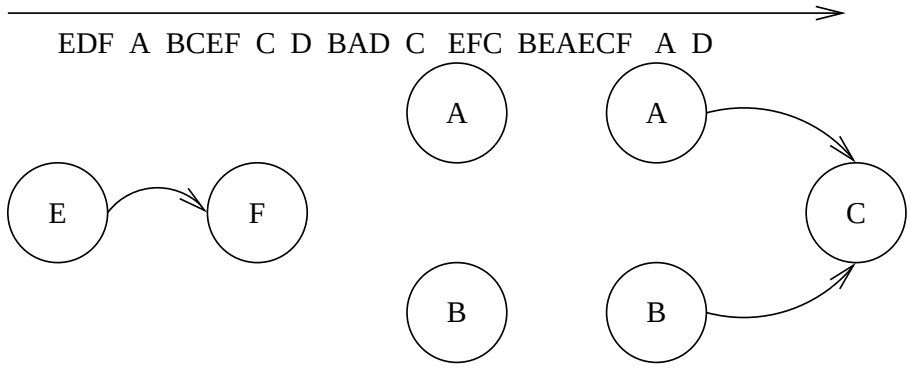
kommen können. Damit dies möglich ist, müssen jedoch ein paar Voraussetzungen für die jeweilige Problemstellung erfüllt sein:

- das Problem muss als Konjuktion ($A \bigwedge B \bigwedge \ldots$) darstellbar sein
- Monotonität (z.B. "wenn Itemset A häufig nicht, kann auch kein Itemset, in dem A vorkommt, häufig sein")
- es muss eine Möglichkeit geben, eine Reihe schnell auf ein Muster zu überprüfen
- nicht zu viele Muster im Datensatz

Solche Erweiterungen auf andere Problemstellungen werden im Folgenden vorgestellt

4.1 Episoden in Sequenzen finden

Bei dieser Problemstellung geht es darum, häufig auftretende Episoden von Ereignissen in Ereignissequenzen zu finden. Solche Ereignisse können beispielsweise Alarme in Telekommunikationsnetzwerken sein.



Beim Finden von Episoden betrachtet man ein bestimmtes Zeitfenster der "Breite" w aus einer potentiell endlosen Sequenz und sucht alle Episoden, die mit einer bestimmten Mindesthäufigkeit min_fr auftreten. Die grundlegende Idee ist wie beim Finden von häufigen Mustern: Man sucht erst nach einfachen Episoden und dann schrittweise nach immer größeren Episoden, die aus diesen einfachen Episoden aufgebaut sind.

4.1.1 Definitionen und Algorithmus

Zum Arbeiten mit Episoden werden zunächst folgende Definitionen festgelegt:

Ereignissequenz S: eine Folge von Tupeln (e, t), wobei e ein Ereignistyp und t die Zeit des Auftretens des Ereignisses ist.

Episode α: eine partielle Ordnung auf Ereignistypen

Fensterbreite w: die Breite eines Ausschnitts von Ereignissen aus einer Ereignissequenz S

Häufigkeit $fr(\alpha)$: der Anteil der Ausschnitte der Breite w aus S, die α enthalten

Subepisode: Eine Episode β ist Subepisode einer Episode α ($\beta \prec \alpha$), falls alle Ereignisse unf Relationen, die in β enthalten sind, auch in α vorkommen und $\alpha \neq \beta$.

Einen an APriori angelehnten Algorithmus kann man als Pseudo- Code dann folgendermaßen aufschreiben:

```
L=1
C_l= alle Episoden a der Lnge 1;
While (C_l not empty)
{
  L_l = alle Episoden a aus Cl mit fr(a, s, win) >= min_fr;
  l = l + 1;
  Cl = alle Episoden a der Laenge l, deren
  saemtliche Subepisoden b in L|b| enthalten sind;
}
```

Mit diesne einfachen Mitteln ist es bereits möglich, Episoden aus Ereignis-Sequenzen zu finden. Oft will man jedoch komplexere Bedingungen an die zu findenden Episoden stellen. Dazu mehr im nächsten Abschnitt.

4.1.2 Generalized Episodes

Mit dem vorgestellten Modell zum Finden von Episoden erlaubt nur das Finden von Episoden mit dem Kriterium, dass die Episoden in einem gewissen Zeitfenster auftreten müssen. Damit stößt man jedoch recht schnell an Grenzen, wenn man komplexere zeitliche und nicht-zeitliche Zusammenhänge zwischen Ereignissen betrachten will.
Wenn man beispielsweise mit Zugriffs- Statistiken einer Website arbeitet,

interessiert unter Umständen z.B. *wer* in *welchen zeitlichen Abständen* auf *welche Seiten* zugegriffen hat.

Man erweitert deswegen die bisherige Definition von Ereignissen so, dass ein Ereignis Attribute haben kann - eines dieser Attribute ist dabei immer der Zeitpunkt des Auftretens, andere können z.B. sein, auf welche Seite zugegriffen wurde oder von welchem Host aus.
Eine Episode kann dann z.B. von der Form $x.page = p_1 \bigwedge y.page = p_2 \bigwedge x.host = y.host$, was so viel aussagt, dass auf die beiden Seiten p_1 und p_2 vom selben Host aus zugegriffen wurde.

Bei der Modellierung der zeitlichen Abhängigkeiten wird außerdem noch folgende Erweiterung vorgenommen: Es können Regeln formuliert werden, die ausdrücken, dass, wenn Ereignisse mit bestimmten Attributen innerhaln einer gewissen Zeit t_1 auftreten, andere Ereignisse innerhalb einer Zeit t_2 auch auftreten. Eine solche Regel ist dann von der Form $P[t_1] \Rightarrow Q[t_2]$ ein Beispiel könnte dann folgendermaßen aussehen: $x.page = p_1 \bigwedge y.page = p_2[60] \Rightarrow x.page = p_1 \bigwedge y.page = p_2 \bigwedge z.page = p_3[120]$. Diese Modellierung der zeitlichen Abhängigkeiten ist nicht nur flexibler, sondern erlaubt auch eine einfachere Neuberechnung der Konfidenz für verschiedene Zeitgrenzen.

Ganz konkret sind "Generalized Episodes" folgendermaßen definiert:

Ereignis e**:** ein Tupel von Attributen $(a_1, a_2, \ldots, t)$, wobei die t der Zeitpunkt und die a_i's sonstige Attribute sind.

Episode P: eine Konjunktion von Prädikaten ψ über Ereignissen

Prädikat ψ**:** ist von einer der Formen

- $x.t \leq y.t$
- $x.a$ (unäres Prädikat, z.B. Testen auf Gleichheit mit Konstante)
- $x.a \circ x.b$ (binäres Prädikat wie z.B. Testen auf Gleichheit)

x und y sind dabei Ereignisse mit Attributen a und b

Episodenregel: $P[t_1] \Rightarrow Q[t_2]$ wobei P und Q Episoden sind und t_1 und t_2 als Zahlenwerte Zeiträume darstellen

Das Suchen von Episoden läuft dann prinzipiell recht ähnlich zu bereits vorgestellten Algorithmen: Von einfachen zu komplexen Episoden.
Ein Problem ist, dass mit der neuen Definition das Finden von beliebigen Episoden P aus einerSequenz S NP-vollständig ist! Wenn man sich aber auf "einfache" Episoden (d.h., die keine binären Prädikate enthält), ist das Problem in einer Zeit der Größenordnung $|P||S|$ lösbar. Weiterhin hat sich herausgestellt, dass auch bei beliebigen Episoden das Problem trotz prinzipieller NP-Vollständigkeit *in der Regel* recht schnell gelöst werden kann.

4.2 Verallgemeinerung der Frequent Pattern Discovery auf andere Problemstellungen

Eine Erweiterung der Methoden der Frequent Pattern Discovery wurde im vorhergehenden Abschnitt erläutert. In diesem Abschnitt wird es nun darum gehen, die Methodik der Frequent Pattern Discovery so zu verallgemeinern, dass sie auf viele verschiedene Probleme anwendbar ist, bei denen andere Kriterien als nur die Häufigkeit von Mustern ins Spiel kommen.

Andere Kriterien als nur der Rückhalt und die Konfidenz einer Regel können sogar notwendig werden - z.B. bei einer Erweiterung der Items von reinen Ja/Nein-Aussagen (etwas anderes ist z.B. "Element a ist in Menge B entahalten" ja nicht) auf reelle Zahlen. Z.B. könnten aus einer Datenbank folgende Regeln destilliert worden sein:

$\alpha : Alter > 40 \Rightarrow Einkommen > 62755$ mit Wahrscheinlichkeit 0,34)
$\beta : Alter > 41 \Rightarrow Einkommen > 62855$ mit Wahrscheinlichkeit 0,33)

Diese Regeln sagen fast das selbe aus und können durchaus beide häufig sein - und von solchen fast gleich lautenden Regeln können potentiell eine große Menge gefunden werden, was es erschwert, die sinnvolle Informationen aus der Flut der Regeln abzulesen und außerdem den Aufwand für die Regelsuche enorm steigern kann.

Wie also können bestehende Algorithmen sinnvoll auf andere Kriterien verallgemeinert werden? Bei der Suche nach Mustern wurden bei der Frequent Pattern Discovery bisher alle Muster als Ergebnisse ausgegeben, die das Kriterium Häufigkeit erfüllen und die selben Muster wurden anschließend

auch auf Spezialisierungen untersucht, die als Ergebnis in Frage kämen,
Davon wird jetzt insofern abgewichen, dass das Kriterium, ob ein Muster ausgegeben werden soll, davon abweichen kann, ob ein Muster auf interessante Spezialisierungen weiteruntersucht werden soll: Das Kriterium für die Ausgabe als Ergebnis nennt man Interessantheit, das Kriterium für die Weiteruntersuchung Potential.
Der Pseudo- code für den verallgemeinerten Algorithmus sieht dann folgendermaßen aus:

```
C = allgemeinstes Muster;
While (C not empty)
  E = alle geeigneten Spezialisierungen der Elemente von C;
  For (each q in E)
  {
    If (q erfllt Interessantheits- Kriterium)
    {
      Gib q als ein Ergebnis aus;
    }
    If (q erfllt Potential- Kriterium NICHT)
    {
      lsche q aus E;
    }
  }
  Filtere E nach weiteren Kriterien;
  C = E;
}
```

Mit diesem Algorithmus kann man nun durch Wahl entsprechender Kriterien Interessantheit und Potential sowie durch das Filtern nach weiteren Kriterien verschiedene Muster-Such-Algorithmen nach verschiedenen Arten von Mustern umsetzen.
Beispiele dafür sind z.B.

Frequent Pattern Discovery: Von der Verallgemeinerung zurück zum Ausgangsalgorithmus kommt man, indem man als Muster Itemsets wählt und die Kriterien Interessantheit und Potential beide als die Häufigkeit eines Itemsets definiert.

Suche nach signifikantesten Regeln: Muster sind vonm der Form $\alpha_1 \bigwedge \ldots \bigwedge \alpha_n \Rightarrow \beta$, wobei *alpha* und β Bedingungen der Form $X = c$, $X < c$ oder $X > c$ sein können. Die Interesantheit wird als die statistische Signifikanz definiert,

das Potential wird immer auf wahr gesetzt. Die Regeln werden dann weiter gefiltert, indem nur die K mit der höchsten Signifikanz ausgewählt werden.

4.3 Interessantheitskriterien

Nachdem im vorigen Abschnitt die Mustersuche auf andere Kriterien ausgedehnt wurde, werden jetzt einige solche Interessantheits- Kriterien vorgestellt.

Diese kann man grundsätzlich in zwei Gruppen unterteilen:

Kriterien aus Anwendungswissen: Damit können oft gute Ergebnisse erzielt werden – sind aber natürlich auch auf das jeweilige Anwedungsgebiet beschränkt und nicht allgemein anwedbar oder definierbar.

Statistische Signifikanz Sind ohne Wissen über Natur und Anwendungsgebiet der Daten anwendbar und können ebenfalls gute Dienste leisten. Im folgenden Abschnitt wird auf diese Art von Kriterien noch etwas näher eingegangen.

4.3.1 Statistische Maße für Interessantheit

Über die Relevanz von gefundenen Regeln über eine Menge von Daten kann oft schon durch statistische Abwägungen gute Aussagen machen. Solche statistischen Signifikanzmaße für Regeln der Form $\theta \Rightarrow \psi$ arbeiten auf einigen aus den Daten errechenbaren Häufigkeiten. Diese sind in der unten abgebildeten Kontingenztabelle zusammengefasst:

	phi	$\neg\psi$
θ	$fr(\theta \wedge \psi$	$fr(\theta \wedge \neg\psi$
$\neg\theta$	$fr(\neg\theta \wedge \psi$	$fr(\neg\theta \wedge \neg\psi$

Die meisten "vernünftigen" solchen statistischen Maße kommen zu relativ ähnlichen Ergebnissen, was auch daran liegt, dass Maße vernünftigerweise bestimmte Kriterien erfüllen wie z.B., dass eine Regel, die bei gleicher Konfidenz häufiger auftritt als eine andere, höher bewertet werden soll.
Ein solches Maßist das J-Maß, das folgendermaßen definiert ist:

$$J(\theta \Rightarrow \psi) = p(\theta)(p(\psi|\theta) log \frac{p(\psi|\theta)}{p(\psi)} + (1 - p(\psi|\theta)) log \frac{1-p(\psi|\theta)}{1-p(\psi)})$$

Für $p(\psi|\theta)$ setzt man dabei die Konfidenz der Regel $\theta \Rightarrow \psi$ ein, für $p(\theta)$ und $p(\psi)$ die Häufigkeiten von θ und ψ. Man benutzt also die relativen Häufigkeiten als Abschätzungen der Wahrscheinlichkeiten.
Allgemein kann man sagen, dass diese Formel Regeln dann hoch bewertet, wenn die linke Seite der Regel großen Einfluss auf die rechte Seite hat - wenn sich also $p(\psi|\theta)$ stark von $p(\psi)$ unterscheidet.

5 Erzeugung globaler Modelle aus lokalen Regeln

Die Regeln, die man mit den bisher kennengelernten Methoden ermitteln kann, geben keine auskunft über die Gesamtstruktur der Daten, sondern sind immer nur für einen Teil der Daten gültig – sie stellen also auch kein Modell für die Gesamtdaten dar.
Falls man aber Voraussagen machen oder neue Daten einordnen will, benötigt man ein Modell der Daten.Im folgenden werden zwei Methoden skizziert, aus den gefundenen Regeln ein Modell zu generieren.

5.1 Klassifizierung durch Regelsatz

Eine Möglichkeit ist dabei, für verschiedene Variablen Klassifizierungsregeln aufzustellen – um den Wert einer Variable B vorherzusagen, wählt man diejenigen Regeln aus, die von der Form $\theta \Rightarrow B = x$ (wobei θ ein Itemset und x ein möglicher Wert von B ist) sind und ordnet sie in einer sortierten Liste an - einer Decision List.
Die Sortierung der Regeln zu finden, die das beste Ergebnis liefert (wäre dabei allerdings zu aufwendig (würde erfordern, alle Möglichkeiten durchzuprobieren). Gute Annäherungen an die optimale Lösung kann man jedoch finden, indem man das Problem als“Weighted Set Cover”-Problem auffasst und per Greedy- Suche nach einem guten Modell sucht.

5.2 Modell als Wahrscheinlichkeitsverteilung

Eine andere Möglichkeit ist, die Häufigkeit von Mustern als Ausdruck einer den Daten inhärenten Wahrscheinlichkeitsverteilung aufzufassen und zu versuchen, eine Wahrscheinlichkeitsverteilung zu generieren, die die beobachteten Häufigkeiten möglichst gut annähert.

Der Algorithmus startet dabei mit einer Zufallsverteilung über den Variablen und geht dann für jedes Muster nacheinander wie folgt vor:

1. Summiere fr jedes Muster die Wahrscheinlichkeiten laut Verteilung ber alle Zustände, die das Muster wahr machen

2. skaliere diese Wahrscheinlichkeiten so, dass die Summe der Wahrscheinlichkeiten laut Verteilung die Häufigkeitsverteilung widerspiegeln.

Diese beiden Schritte werden so lange für alle Muster wiederholt, bis die erzeugte Wahrscheinlichkeits-Verteilung mit den Häufigkeiten der Muster zusammenpasst.

6 Résumée

Nachdem die vorangegangenen Kapitel hoffentlich ein grobes Verständnis über die Problemstellungen, Techniken und einige anwendungsgebiete vermittelt haben, abschließend noch ein paar Bemerkungen zu Schwächen und Stärken der vorgestellten Verfahren:

Zu den Stärken der Verfahren gehört sicherlich, dass die Regeln nach dem Schema "Falls A und B, dann auch C" für den Menschen gut verständlich und aus dem Alltag vertraut sind, während bei Ergebnissen anderer Verfahren zwar Strukturen ersichtlich werden, nicht jedoch das Warum und Wieso.

Eine weitere Stärke ist auch, dass damit auch "feinere" Regelmäßigkeiten erkannt werden können, die bei anderen Verfahren wie beispielsweise Clustering unter den Tisch fallen würden.

Eine Schwċhe oder besser gesagt Einschränkung ist, dass die Verfahren ursprünglich für mit dünnbesetzte Datensätzen im Sinn entworfen wurden und bei dichter besetzten Daten viele dieser Algorithmen auf Laufzeitprobleme stoßen können.

Dadurch, dass die vorgstellten Algorithmen zunächst kein globales Datenmodell erzeugen, sondern nur Regeln, die für Teilmengen der Daten Relevanz haben, sind sie für Vorhersage schwierig einzusetzen – auch wenn, wie in Abschnitt 5 geschildert, durchaus Möglichkeiten dazu existieren,

Letztendlich muss man sich anhand der Charakteristika des Anwendungsgebietes bzw. der Problemstellung entscheiden, ob dieser ansatz oder doch eher ein anderer zur Lösung geeignet ist.

Detailliertere Informationen zu den vorgestellten Methoden finden sich in der im folgenden Abschnitt aufgelisteten Literatur.

7 Verwendete Literatur

"Principles Of Data Mining", Kapitel 13 – David Hand, Heikki Mannila, Padhraic Smyth
Themen: Grrundlegendes über Frequent Pattern Discovery (Frequent Itemsets, Assoziationsregeln), APriori, Frequent Episodes, Verallgemeinerung von Frequent Pattern Discovery, Erzeugung globaler Modelle

"Efficiently Mining Long Patterns from Databases" – Roberto J. Bayardo Jr.
Themen: Vorstellung und Erklärung des MaxMiner-Algorithmus
URL: http://www.almaden.ibm.com/software/quest/Publications/papers/sigmod98_max.p

"Levelwise Search and borders of theories in knowledge dicovery" – Heikki Mannila, Hannu Toivonen
Themen: Vertiefung zu Suchalgorithmen und Frquent Sets (bzw. Sentences)
URL: http://www.cs.helsinki.fi/TR/C-1997/8/C-1997-8.ps.gz

"Discovering generalized episodes using minimal occurences" – Heikki Mannila, Hannu Toivonen
Themen: Generalized Episodes
URL: http://arbor.ee.ntu.edu.tw/ netdb/course/datamining/papers/manndg96.pdf