Data Mining

Assoziationsregeln

Dr. Hanna Köpcke

Wintersemester 2020

Abteilung Datenbanken, Universität Leipzig
http://dbs.uni-leipzig.de

Übersicht

Hochdimensionale Daten

- Clustering
- Dimensions-reduktion
- Empfehlungs-systeme
- Assoziations-regeln
- Locality Sensitive Hashing
- Supervised ML

Graphdaten

- Community Detection
- PageRank
- Web Spam

Datenströme

- Windowing
- Filtern
- Momente
- Web Advertising

Inhaltsverzeichnis

- **Einleitung**
- **A-Priori Algorithmus**
- **PCY Algorithmus**
- **Algorithmen mit weniger Durchläufen**
  - Zufallsstichprobe
  - SON Algorithmus
  - Algorithmus von Toivonen

Motivation

- *Warenkorbanalyse* (Market Basket Analysis): Analyse des Kaufverhaltens
- **Datengrundlage**:
  - Große Menge an **Elementen** und große Menge an **Warenkörben**
  - Zuordnungen von kleinen Mengen an Elementen zu Warenkörben
- **Beispiel**: Kaufverhalten im Supermarkt
  - Welche Produkte werden von hinreichend vielen Leuten (nicht) zusammen gekauft?
  - Klassische (Assoziations-)Regeln:
    - Wenn jemand Bratwürste kauft, dann auch Senf.
    - Wenn jemand Coca Cola kauft, dann nicht Pepsi.
    - Wenn jemand Windeln kauft, dann auch Bier.

Warenkörbe
Brot, Cola, Milch
Milch, Windeln
Bier, Cola, Windeln, Milch
Bier, Brot, Windeln, Milch
Cola, Windeln, Milch, Bier

→

Assoziationsregeln:

{Milch} → {Cola}
{Windeln, Milch} → {Bier}

Anwendungen

- Kaufverhalten für Werbung, Treueprogramme, Store-Design, Rabattpläne oder den „Querverkauf“ von Produkten
- Produktempfehlungen:
  - „Kunden, die diesen Artikel gekauft haben, kauften auch …“
  - „Kunden, die diesen Film angesehen haben, haben auch angesehen …“
- Thematisch verwandte Konzepte: Wörter (Elemente) und Dokumente (Warenkörbe)
- Plagiate: Dokumente (Elemente) und Sätze (Warenkörbe)
- Nebenwirkungen von bestimmten Kombination von Medikamenten
  - Elemente: Nebenwirkung und Medikament
  - Warenkörbe: Patienten

Häufige Elementmengen

- *Elementmenge* = Teilmenge der Elemente
- **Support** einer Elementmenge I $(\text{sup}(I))$: Anteil der Warenkörbe, welche alle Elemente aus I enthalten
- Gegeben eines Schwellenwerts s, eine Elementmenge I wird als **Häufige Elementmenge** bezeichnet, falls $\text{sup}(I) \geq s$

Warenkörbe
Brot, Cola, Milch
Milch, Windeln
Bier, Cola, Windeln, Milch
Bier, Brot, Windeln, Milch
Cola, Windeln, Milch, Bier

Häufige Mengen ($s = 0.6$)	sup
{Milch}	1.0
{Windeln}, {Milch, Windeln}	0.8
{Cola}, {Bier}, {Bier, Milch}, {Cola, Milch}, {Windeln, Bier}, {Bier, Windeln, Milch}	0.6

Assoziationsregeln

- **Assoziationsregel**: Wenn-Dann-Regel zu den Inhalten der Warenkörbe
  - *Form*: $\{i_1, i_2, \ldots, i_k\} \rightarrow j$
  - *Interpretation*: Falls ein Korb die Elemente $i_1, i_2, \ldots, i_k$ enthält, dann enthält er mit hoher Wahrscheinlichkeit das Element j

- Auswahl einer Regeln über deren **Confidence**:

$$\mathrm{conf}(\{i_1, i_2, \ldots, i_k\} \rightarrow j) = \frac{\sup(\{i_1, i_2, \ldots, i_k, j\})}{\sup(\{i_1, i_2, \ldots, i_k\})}$$

- Beispiel: falls $conf(\{i_1, \ldots, i_k\} \rightarrow j) > 0.8$, dann kommt in über 80% aller Warenkörbe mit den Elementen $i_1, \ldots, i_k$ auch das Element j vor

Beispiel

Warenkörbe
Brot, Cola, Milch
Milch, Windeln
Bier, Cola, Windeln, Milch
Bier, Brot, Windeln, Milch
Cola, Windeln, Milch, Bier

Häufige Mengen ($s = 0.6$)	sup
{Milch}	1.0
{Windeln}, {Milch, Windeln}	0.8
{Cola}, {Bier}, {Bier, Milch}, {Cola, Milch}, {Windeln, Bier}, {Bier, Windeln, Milch}	0.6

Regeln	conf
{Cola} $\longrightarrow$ {Milch}, {Bier} $\longrightarrow$ {Windeln}, {Bier} $\longrightarrow$ {Milch}, {Windeln} $\longrightarrow$ {Milch}, {Bier,Windeln} $\longrightarrow$ {Milch}, {Bier, Milch} $\longrightarrow$ Windeln	1.0
{Milch} $\longrightarrow$ {Windeln}	0.8
{Windeln} $\longrightarrow$ {Bier}, {Windeln, Milch} $\longrightarrow$ Bier	0.75
{Milch} $\longrightarrow$ {Cola}, {Milch} $\longrightarrow$ {Bier}	0.6

Die Suche nach Assoziationsregeln

- **Problem**: Finde alle Assoziationsregeln $I \rightarrow j$ mit $\mathrm{sup}(I \cup \{j\}) \geq s$ und $\mathrm{conf}(I \rightarrow j) \geq c$

- **Schwierigkeit**: Suche nach häufigen Elementmengen mit $sup(I) \geq s$ bzw. die Berechnung des Supports aller möglichen Elementmengen

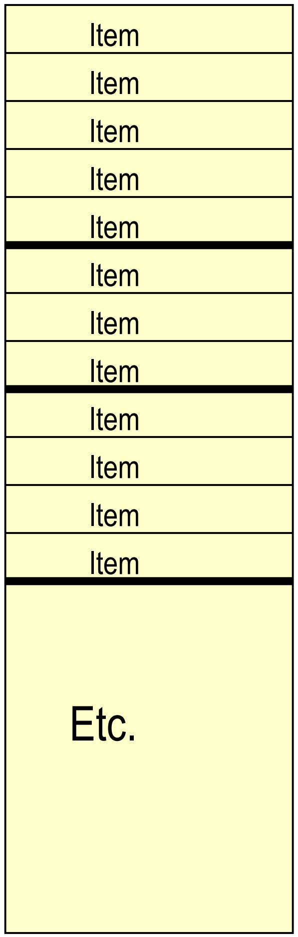
Konstruktion der Assoziationsregeln aus häufigen Elementmengen:

- Für jede häufige Elementmenge I und jedes Element j aus I
  - Erstelle die Regel $I \backslash \{j\} \rightarrow j$
  - Ausgabe der Regel, falls $\mathrm{conf}(I \backslash \{j\} \rightarrow j) = \frac{sup(I)}{sup(I \backslash \{j\})} \geq c$
  - (da I häufig ist, ist auch $I \backslash \{j\}$ häufig; somit wurde sowohl sup(I) als auch sup($I \backslash \{j\}$) schon berechnet)

Suche nach **häufigen Elementmengen**

- Annahme, dass Daten in Dateisystem auf Festplatte gespeichert
- Sortiert nach Warenkörben
- Wenige Elemente pro Korb: Aufwand der Generierung aller Teilmengen eines Korbs ist relativ gering

- Entscheidend: **Anzahl der Warenkörbe**, insb. wenn diese nicht vollständig in Hauptspeicher passen
  - **1. Problem:** Speichern der Zwischenergebnisse (Häufigkeiten der Teilmengen) in Hauptspeicher
  - **2. Problem:** Evtl. mehrere Festplattendurchläufe nötig (Lesen aller Warenkörbe von Festplatte)



Problem: Speichern der Zwischenergebnisse

- Beispiel: Bestimmen der häufigen Paare
- Zählen im Hauptspeicher ist nicht für große Dimensionen geeignet
  - Angenommen n verschiedene Elemente und Speichern der Zähler (Integer: 4 Byte) als Dreiecksmatrix = Array der Form {1,2}, {1,3},…, {1,n}, {2,3}, {2,4},…,{2,n}, {3,4},…
  - Speicherplatz für $\binom{n}{2} = \frac{n(n-1)}{2} \approx \frac{n^2}{2}$ Zähler: $2n^2$ Byte
  - Bei 32 GB Hauptspeicher: $n < 2^{17} \approx 130\,000$
  - Damit würde weder Walmart (17 Millionen Produkte) noch Amazon (500 Millionen Produkte) auskommen
- Weniger Speicherplatz notwendig, falls nicht alle Paare auftreten
  - Speichern als *spärlich besetzte Matrix*: Liste von Tripeln $[i, j, c]$ wobei i und j die Elemente bezeichnen und c den Zähler: 12 Byte pro Paar
  - Weniger Speicherplatz, falls maximal 1/3 der Paare auftreten

Inhaltsverzeichnis

A-Priori Algorithmus

- Wichtige Eigenschaft der häufigen Elementmengen: **Monotonie**

$$J \subseteq I \Rightarrow \sup(I) \leq \sup(J)$$

- *Gegenrichtung*: Falls eine Menge J keine häufige Elementmenge ist, dann ist keine Menge I, welche J enthält, eine häufige Elementmenge

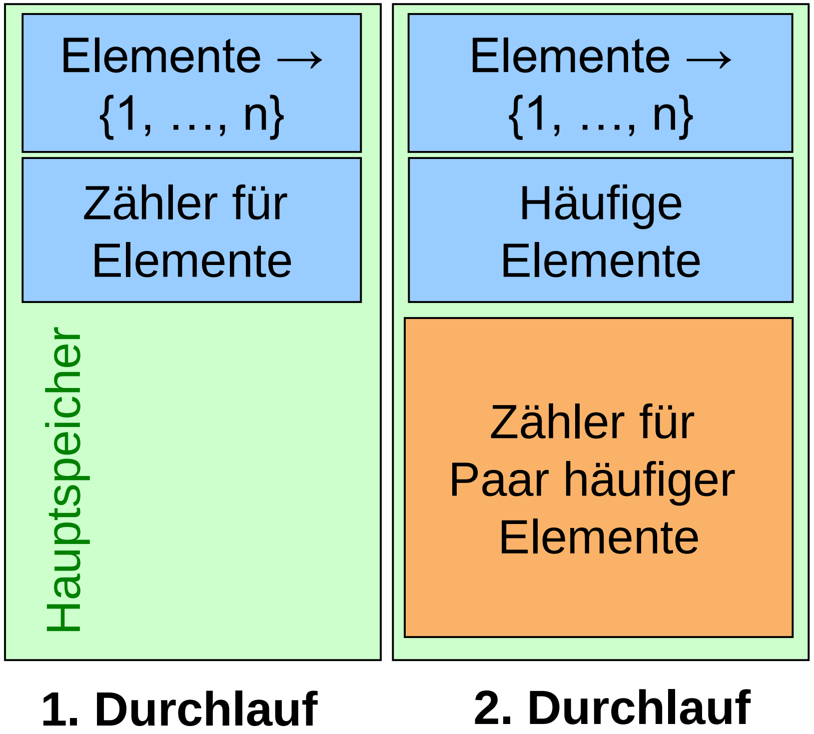
- **A-Priori Algorithmus**: Begrenzung des benötigten (Haupt-) Speicherplatzes über zweifaches Einlesen aller Daten
  - 1. Durchlauf: Zählen aller Elemente (einelementige Mengen) und Auswahl der häufigen Elemente über Schwellenwert s
  - 2. Durchlauf: Zählen eines vorkommenden Paares nur dann, wenn beide Elemente häufig sind

A-Priori Algorithmus: Details

- Tabelle: Eindeutige Abbildung der Elemente auf die Zahlen 1, …, n

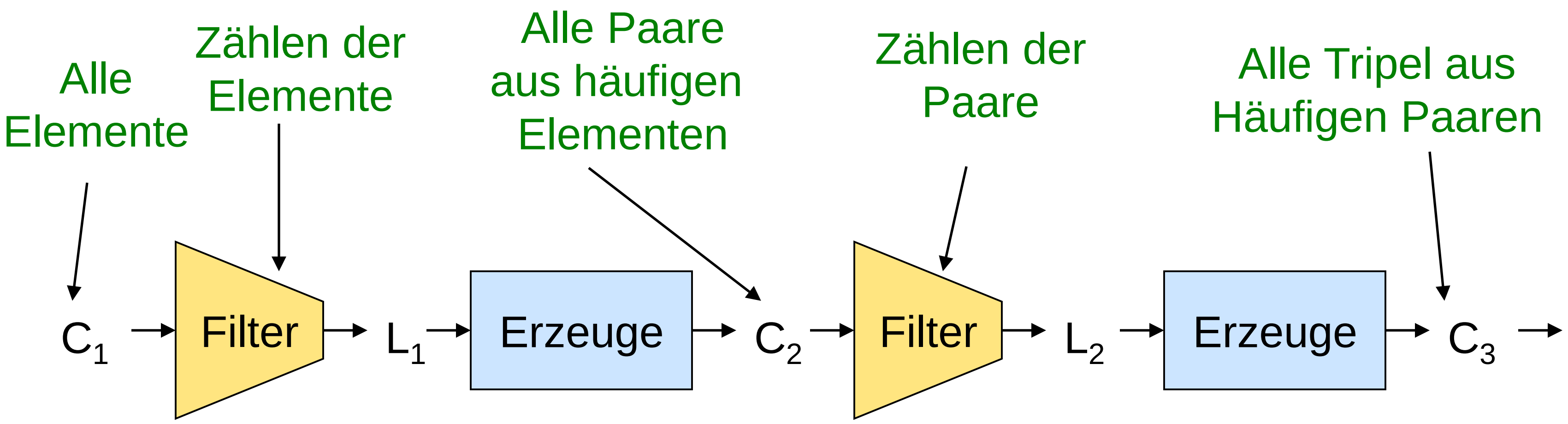
- **1. Durchlauf**: Zähler für Elemente ist einfaches Array der Länge n
- Danach: Neues Array der gleichen Länge, aber Zuordnung der häufigen Element zu neuer Nummerierung 1, …, m und seltene Elemente auf 0

- **2. Durchlauf**: Speicherung der Paare häufiger Elemente über Dreiecksmatrix/ spärlich besetzte Matrix



Häufige Tripel etc.

- Weiterer Durchlauf für jede Menge der Größe k
- Menge C_k von Kandidaten
  - Kandidat = Menge der Größe k, die aufgrund der Informationen über Mengen der Größe k-1 häufig sein könnten
  - Für einen Kandidat gilt: alle (k-1)-elementigen Teilmengen müssen häufig sein
- Menge L_k der häufigen Mengen der Größe k



Beispiel

$$C_1 = \{\{\mathrm{b}\}, \{\mathrm{c}\}, \{\mathrm{j}\}, \{\mathrm{m}\}, \{\mathrm{n}\}, \{\mathrm{p}\}\}$$

$$\downarrow$$

$$L_1 = \{\{\mathrm{b}\}, \{\mathrm{c}\}, \{\mathrm{j}\}, \{\mathrm{m}\}\}$$

$$\downarrow$$

$$C_2 = \{\{\mathrm{b}, \mathrm{c}\}, \{\mathrm{b}, \mathrm{j}\}, \{\mathrm{b}, \mathrm{m}\}, \{\mathrm{c}, \mathrm{j}\}, \{\mathrm{c}, \mathrm{m}\}, \{\mathrm{j}, \mathrm{m}\}\}$$

$$\downarrow$$

$$L_2 = \{\{\mathrm{b}, \mathrm{c}\}, \{\mathrm{b}, \mathrm{m}\}, \{\mathrm{c}, \mathrm{j}\}, \{\mathrm{c}, \mathrm{m}\}\}$$

$$\downarrow$$

$$C_3 = \{\{\mathrm{b}, \mathrm{c}, \mathrm{m}\}\}$$

$$\downarrow$$

$$L_3 = \{\}$$

Inhaltsverzeichnis

Park-Chen-Yu (PCY) Algorithmus

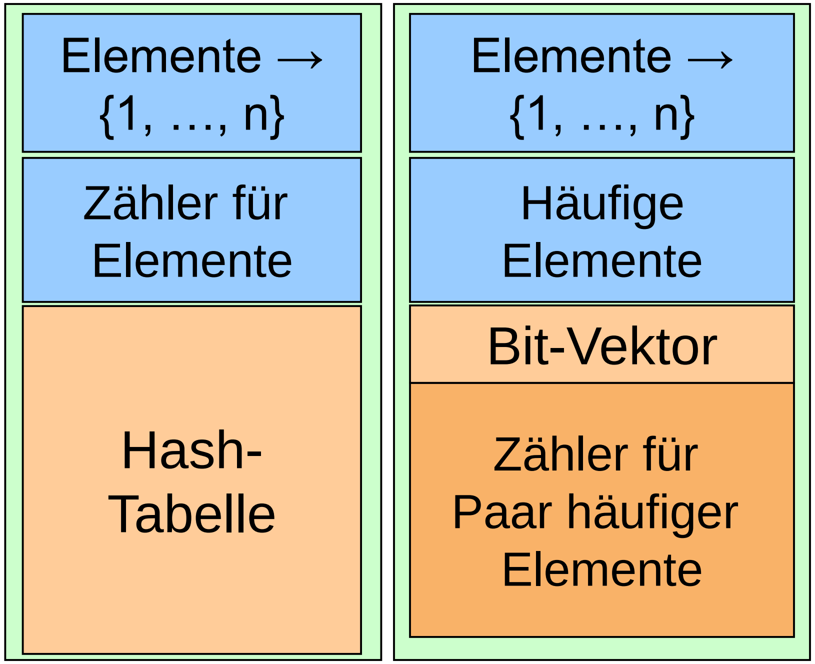
- In einigen Fällen könnte Hauptspeicher für den 2. Durchlauf des A-Priori Algorithmus nicht ausreichen
- *Idee*: Verwende den restlichen Speicherplatz im 1. Durchlauf, um die Kandidatenmenge weiter zu reduzieren

PCY-Algorithmus:

- Verwendung einer Hash-Funktion mit möglichst vielen Buckets und einem Array aus Zählern (ein Zähler pro Bucket)
- Für jeden Warenkorb:
  - Für jedes Element des Warenkorbs: Addiere 1 zu dem Zähler des Elements
  - **Für jedes Paar von Elementen:**
    - **Anwendung der Hash-Funktion auf Paar**
    - **Addiere 1 zu dem Zähler des Bucket**

PCY Algorithmus

- Beobachtungen:
  - Falls ein Bucket ein häufiges Paar enthält (Support größer als ein Schwellenwert s), dann ist der Anteil $\left(\frac{Zähler}{Anzahl\ der\ Warenkörbe}\right)$ dieses Bucket größer als s
  - Auch seltene Paare (sup $< s$) können in Buckets mit einem Anteil $> s$ vorkommen
  - **Buckets mit einem Anteil** $< s$ **können keine häufigen Paare** (sup $> s$) **enthalten**
- *Folgerung*: Alle Paare aus Buckets mit Anteil $< s$ müssen im 2. Durchlauf nicht betrachtet werden
- Für 2. Durchlauf:
  - Ersetze Hash-Tabelle durch Bit-Vektor: 1 bedeutet, dass Anteil des Bucket größer als s ist
  - Bit-Vektor benötigt nur $\frac{1}{32}$ des Speichers der Hash-Tabelle (Integer: 4 Byte)



1. Durchlauf **2. Durchlauf**

PCY Algorithmus

- **2. Durchlauf**: Zähle ein Paar $\{i, j\}$ genau dann, wenn
  - beide Element i und j häufig sind und
  - $\{i, j\}$ über die Hash-Funktion auf ein Bucket mit Wert 1 im Bit-Vektor abgebildet wird

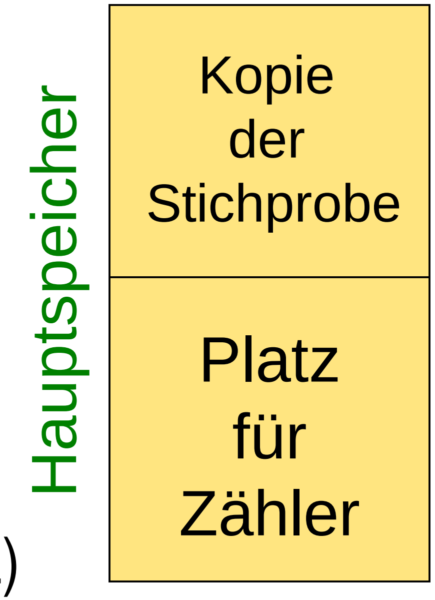
- Anmerkungen:
  - **Große Anzahl an Buckets notwendig**
  - Die Buckets der Hash-Tabelle benötigen nur wenige Bytes: man muss nur bis $s \cdot (\text{Anzahl der Warenkörbe})$ zählen → Je nach Hauptspeicher: große Anzahl an Buckets möglich
  - PCY sollte 2/3 der Kandidaten eliminieren, damit Liste von Tripeln (spärlich besetzte Matrix) im 2. Durchlauf verwendbar und PCY tatsächlich effizienter als A-Priori
  - Falls Hauptspeicher für 2. Durchlauf noch nicht ausreichend: weitere Einschränkung der Kandidatenmenge über erneuten Durchlauf möglich (**Multistage PCY**)
  - In manchen Fällen ist es effizient, zwei verschiedene Hash-Funktionen anzuwenden und daraus zwei kleinere Bit-Vektoren zu erzeugen (**Multihash PCY**)

Inhaltsverzeichnis

- **Einleitung**
- **A-Priori Algorithmus**
- **PCY Algorithmus**
- **Algorithmen mit weniger Durchläufen**
  - Zufallsstichprobe
  - SON Algorithmus
  - Algorithmus von Toivonen

Zufallsstichprobe

- Ziehe Zufallsstichprobe, so dass alle Warenkörbe der Stichprobe und die benötigten Zähler in den Hauptspeicher passen
- Anwendung von A-Priori oder PCY im Hauptspeicher
- Nur ein Festplattendurchlauf für die Suche nach den häufigen Elementmengen **aller Größen**



- Vermeidung von **False Positives**: Verifiziere die über die Stichprobe ausgewählten häufigen Elementmengen durch einen *weiteren Durchlauf* (benötigt weniger Speicher, da weniger Kandidaten)
- Reduzierung der **False Negatives**: Kleineren Schwellenwert, z.B. $0.9 \cdot s$ (benötigt mehr Speicherplatz)

SON Algorithmus

- Savasere-Omiecinski-Navathe (SON) Algorithmus:
  - Lade wiederholt eine feste Anzahl an Warenkörben (Chunk) in Hauptspeicher
  - Finde häufige Elementmengen **aller Größen**

- Gesamter Datensatz wird in Teilen durchlaufen und häufige Elementmengen der Chunks werden gespeichert

- **Keine False Negatives**: Eine Elementmenge kann nicht im gesamten Datensatz mit Schwellenwert s häufig vorkommen, ohne in mindestens einem Chunk mit Schwellenwert s häufig vorzukommen

- Aussortieren der **False Positives** über weiteren Durchlauf

Algorithmus von Toivonen

- **1. Durchlauf**:
  - Ziehen einer Zufallsstichprobe, die Arbeit in Hauptspeicher erlaubt
  - Nimm die Elementmengen **aller Größen**, die mit Schwellenwert $0.9 \cdot s$ häufig sind, als *Kandidaten*
  - Erstellen der **Negativen Grenze** zu Kandidaten: eine Menge ist in der Negativen Grenze falls sie kein Kandidat ist, aber alle direkten Untermengen (Untermengen mit genau einem Element weniger) Kandidaten oder die leere Menge sind

- Beispiel: Grundmenge {A,B,C,D,E} wobei {A}, {B}, {C}, {D}, {B,C}, {C,D} häufig in Stichprobe vorkommen
  - Negative Grenze: {E}, {A,B}, {A,C}, {A,D}, {B,D}
  - Die anderen Paare sind nicht in Negativen Grenze, da sie entweder E enthalten oder häufig vorkommen
  - Kein Tripel ist in Negativen Grenze

Algorithmus von Toivonen

- **2. Durchlauf**: Zählen der Kandidaten und der Elemente der Negativen Grenze im gesamten Datensatz
  - Aussortieren der **False Positives**
  - Falls kein Element der Negativen Grenze häufig vorkommt, wurden alle häufigen Elementmengen gefunden (**keine False Negatives**)
  - Falls ein Element der Negativen Grenze häufig vorkommt, muss der Algorithmus mit neuer Zufallsstichprobe wiederholt werden (evtl. mit geringerem Schwellenwert)
- **Satz**: Falls eine Elementmenge S häufig im gesamten Datensatz aber nicht in der Zufallsstichprobe vorkommt, dann enthält die Negative Grenze mind. eine Menge, die häufig im gesamten Datensatz vorkommt.
- *Beweis*:
  - Sei T die kleinste Untermenge von S die nicht häufig in der Stichprobe vorkam, aber jede Teilmenge von T kam häufig in Stichprobe vor
  - Dann ist T in der Negativen Grenze.
  - Da S häufig in gesamter Menge, ist auch T häufig in gesamten Menge (Monotonie)