Frequent Pattern Mining
Bart Goethals
University of Antwerp, Belgium
ADReM research group
http://www.adrem.ua.ac.be/~goethals/
What this talk is about

- One of the most popular problems in computer science!
- [Agrawal, Imielinski, Swami, SIGMOD 1993] 13th most cited across all computer science
- [Agrawal, Srikant, VLDB 1994] 15th most cited across all computer science
- [Goethals, 2003] a nice survey
- several other very interesting papers
Pattern Mining

- Unsupervised learning
- Local (vs. global models)
- Useful for
  - large datasets
  - exploration: « what is this data like? »
  - building global models
- Less suitable for
  - well-studied and understood problem domains
Outline

• Mining association rules
• Algorithms
  - Apriori
  - Eclat
  - FP-growth
• Optimizations and Extensions
• Other pattern types
• General levelwise search
• Other interestingness measures
Back in 1993...

- Find associations between products
- For example: a supermarket
  - which products are frequently bought together?
  - do some products influence the sales of other products?
    e.g. “75% of people who buy beer, also buy chips”
Applications

- Supermarket
  - cross selling
  - product placement
  - special promotions
- Websearch
  - which keywords often occur together in webpages?
- Health care
  - frequent sets of symptoms for a disease
- Prediction
  - associative classifiers
- ...

Universiteit Antwerpen
Applications

• Basically works for all data that can be represented as a set of examples/objects having certain properties
  - patient / symptoms
  - movies / ratings
  - web pages / keywords
  - basket / products
  - ...
Formally

- A **transaction database** is a collection of sets of items (transactions)
- An **itemset** is a set of items
- An **association rule** is an implication of the form \( X = \rightarrow Y \), with \( X \) and \( Y \) itemsets
- **Support Count** (SC) of an itemset \( X \) is the number of transactions that contain \( X \)
- **Support** of \( X \) (also **frequency** of \( X \)) = \( \text{SC}(X)/\text{SC}() \)
- **Support** of an association rule \( X = \rightarrow Y \) equals the support of \( X \cup Y \)
- **Confidence** of an association rule \( X = \rightarrow Y \) = \( \frac{\text{Support}(X = \rightarrow Y)}{\text{Support}(X)} \)
Problem

• Given:
  - a transaction database
  - a minimum support threshold
  - a minimum confidence threshold

• Find all rules $X \Rightarrow Y$ such that:
  - $\text{Support}(X \Rightarrow Y) > \text{minsup}$
    ($X \Rightarrow Y$ is frequent)
  - $\text{Confidence}(X \Rightarrow Y) > \text{minconf}$
    ($X \Rightarrow Y$ is confident)
Example

- minimum support = 2
- minimum confidence = 2/3
- \{\text{shoes}\} \Rightarrow \{\text{socks}\} \text{ is a confident association rule with support} = 0.5, \text{confidence} = 1
- \{\text{socks}\} \Rightarrow \{\text{shoes}\} \text{ is not}
- \text{Sweater can not appear in a rule}

<table>
<thead>
<tr>
<th>Tid</th>
<th>Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>shoes, socks, T-shirt</td>
</tr>
<tr>
<td>2</td>
<td>socks, sweater, pants</td>
</tr>
<tr>
<td>3</td>
<td>T-shirt, pants, socks</td>
</tr>
<tr>
<td>4</td>
<td>shoes, socks</td>
</tr>
</tbody>
</table>
• Solution #1:
  - Generate all possible rules
  - Count their supports and compute confidence
  - INTRACTABLE… (\(3^n\) possible combinations)

• Solution #2:
  - First, find all frequent itemsets
  - Second, split every frequent itemset \(Z\) in two parts \(X\) and \(Y\), such that \(X \Rightarrow Y\) is confident

• Example: \(I = \{A,B,C\}\)
  test rules \(\{A,B\} \Rightarrow \{C\}, \{AC\} \Rightarrow \{B\}, \{B,C\} \Rightarrow \{A\},\)
  \(\{A\} \Rightarrow \{B,C\}, \{B\} \Rightarrow \{A,C\}, \{C\} \Rightarrow \{A,B\}\)
How to find all frequent itemsets?

• Solution #1:
  - Generate all possible itemsets
  - Count their support in DB
  - INTRACTABLE... (2^n possible combinations)
How to find all frequent itemsets?

- **Solution #2:**
  - Apriori
  - Rakesh Agrawal and Srikant Ramakrishnan [VLDB, 1994]
  - Heikki Mannila and Hannu Toivonen [KDD, 1994]
Apriori

- Key observation: (monotonicity)

A subset of a frequent itemset must also be frequent, or,

any superset of an infrequent itemset must also be infrequent!
Apriori

- An itemset is called a **candidate itemset** if all of its subsets are known to be frequent
- Solution:
  Iteratively find frequent itemsets with cardinality from 1 to k (k-itemset)
Example

- Start with small itemsets, only proceed with larger itemset if all subsets are frequent
- \{ A, B, C \} is evaluated after \{A\}, \{B\}, \{C\}, \{A,B\}, \{A,C\}, and \{B,C\}, and only if all these sets are known to be frequent
Level-wise search

minsup = 2

<table>
<thead>
<tr>
<th>Tid</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A, C, D</td>
</tr>
<tr>
<td>2</td>
<td>B, C</td>
</tr>
<tr>
<td>3</td>
<td>A, B, C, D</td>
</tr>
</tbody>
</table>

C1: A B C D

C2: AB AC AD BC BD CD

C3: ACD BCD
The Apriori Algorithm

\( C_k \): candidate itemset of size \( k \)
\( L_k \): frequent itemset of size \( k \)

\[ L_1 = \{ \text{frequent items} \}; \]
\[ \text{for } (k = 1; \ L_k \neq \emptyset; \ k++) \text{ do begin} \]
\[ C_{k+1} = \text{candidates generated from } L_k; \]
\[ \text{for each transaction } t \text{ in database do} \]
\[ \text{increment the count of all candidates in } C_{k+1} \]
\[ \text{that are contained in } t \]
\[ L_{k+1} = \text{candidates in } C_{k+1} \text{ with min\_support} \]
\[ \text{end} \]
\[ \text{return } \cup_k L_k; \]

Universiteit Antwerpen
Candidate Generation

- for all itemsets $X$, $Y$ with $X[\vdash:1]=Y[\vdash:1]$
- $X + Y[-1:]$ is a candidate itemset,
- only if all its subsets are known to be frequent
- note that $\{1,2,3\}$ was not even considered
Example run

TID  | Items  
-----|--------
100  | 1 3 4  
200  | 2 3 5  
300  | 1 2 3 5 
400  | 2 5    

Scan D

C₁

<table>
<thead>
<tr>
<th>itemset</th>
<th>sup.</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1}</td>
<td>2</td>
</tr>
<tr>
<td>{2}</td>
<td>3</td>
</tr>
<tr>
<td>{3}</td>
<td>3</td>
</tr>
<tr>
<td>{4}</td>
<td>1</td>
</tr>
<tr>
<td>{5}</td>
<td>3</td>
</tr>
</tbody>
</table>

L₁

<table>
<thead>
<tr>
<th>itemset</th>
<th>sup.</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1}</td>
<td>2</td>
</tr>
<tr>
<td>{2}</td>
<td>3</td>
</tr>
<tr>
<td>{3}</td>
<td>3</td>
</tr>
<tr>
<td>{5}</td>
<td>3</td>
</tr>
</tbody>
</table>

Scan D

C₂

<table>
<thead>
<tr>
<th>itemset</th>
<th>sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1 2}</td>
<td>1</td>
</tr>
<tr>
<td>{1 3}</td>
<td>2</td>
</tr>
<tr>
<td>{1 5}</td>
<td>1</td>
</tr>
<tr>
<td>{2 3}</td>
<td>2</td>
</tr>
<tr>
<td>{2 5}</td>
<td>3</td>
</tr>
<tr>
<td>{3 5}</td>
<td>2</td>
</tr>
</tbody>
</table>

Scan D

C₃

<table>
<thead>
<tr>
<th>itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>{2 3 5}</td>
</tr>
</tbody>
</table>

Scan D

L₂

<table>
<thead>
<tr>
<th>itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>{2 3 5}</td>
</tr>
</tbody>
</table>

Scan D

C₄

<table>
<thead>
<tr>
<th>itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>{2 3 5}</td>
</tr>
</tbody>
</table>

Scan D

L₃

<table>
<thead>
<tr>
<th>itemset</th>
<th>sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>{2 3 5}</td>
<td>2</td>
</tr>
</tbody>
</table>
Apriori’s main problem

- In every count step we have to do a very costly scan over the complete database.
Optimizations

- **Dynamic Itemset Counting** [Brin et al., 1997]
  - interrupt algorithm after every M transactions and already generate larger candidates if possible

- **Partition** [Savasere et al., 1995]
  - partition database, and mine each part separately (using relative minsup!)
  - Union of all frequent itemsets of all parts are a superset of all frequent itemsets in complete database!
  - Extra pruning step

- **Sampling** [Toivonen, 1995]
  - Run apriori on small sample of DB
  - Correct result
Current Research

• Until today, many researchers still try to find new techniques, and improve Apriori
  - Optimized for sparse/dense data
  - Optimized for many/few items

• Implementation issues are important
  - How to implement the counting step
  - How to read the database
  - How to generate the candidates
  - How to prune the candidates
  - Ordering of items is important!

• For more info: visit http://fimi.cs.helsinki.fi/
What if DB fits in memory?

• Faster counting of supports!

• Two new techniques differ in counting strategy and how the database is represented in memory
  - Eclat [Zaki et al., KDD 1997]
  - FP-growth [Han et al., SIGMOD 2000]
Eclat: tidlist

- For every item, a list of transaction id’s is stored in which the item occurs, denoted by tidlist

- For every itemset, its tidlist equals the intersection of the tidlists of two of its subsets
Eclat: tidlist example

{a}  1 2 3 4 5
{b}  1 3 5 6 7
{a,b}  1 3 5

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{a,b}</td>
</tr>
<tr>
<td>2</td>
<td>{a}</td>
</tr>
<tr>
<td>3</td>
<td>{a,b}</td>
</tr>
<tr>
<td>4</td>
<td>{a}</td>
</tr>
<tr>
<td>5</td>
<td>{a,b}</td>
</tr>
<tr>
<td>6</td>
<td>{b}</td>
</tr>
<tr>
<td>7</td>
<td>{b}</td>
</tr>
</tbody>
</table>
Eclat: algorithm

- In principle Apriori could be used together with intersection based support counting
- Memory usage, however, would blowup!
- Therefore, a depth-first approach is used
Divide and conquer

1. Find all itemsets containing \{a\}
2. Find all itemsets not containing \{a\}
   - For 1. Only transactions containing \{a\} are necessary (\{a\} can be removed)
   => \{a\}-conditional database
   - For 2. \{a\} can be removed from all transactions
   - Apply recursively
1. Get tidlist for each item (DB scan)
2. Tidlist of \{a\} is exactly the list of transactions containing \{a\}
3. Intersect tidlist of \{a\} with the tidlists of all other items, resulting in tidlists of \{a,b\}, \{a,c\}, \{a,d\}, ... = \{a\}-conditional database (if \{a\} removed)
4. Repeat from 1 on \{a\}-conditional database
5. Repeat for all other items
• Database is stored in FP-tree
FP-growth

- Divide and conquer strategy is used
  1. Find all itemsets containing \{a\}
  2. Find all itemsets not containing \{a\}
- For 1. Only transactions containing \{a\} are necessary (\{a\} can be removed)
  => \{a\}-conditional database
- For 2. \{a\} can be removed from all transactions
- Apply recursively
Apriori vs. Eclat vs. FP-growth

• Which is best? Depends on data
• Apriori better for huge databases
• Eclat most of the time better than FP-growth
• Many optimizations exist! (see FIMI)

• FP-growth paper title says: “Mining Frequent Patterns without candidate generation”
• Where did the candidates go?
Some FIMI results
Some FIMI conclusions

- There is no clear winner
- Much depends on implementation details
- Experiments should be reproducible and therefore source code should be available!
Extensions

• Maximal Itemset Mining [Bayardo, 1998]
  - One might not be interested in all frequent itemsets, but only in the maximal ones
  - Optimized algorithms exist

• Closed Itemset Mining [Pasquier et al., 1999]
  - Suppose A => X holds with 100% confidence
  - Then, every itemset containing A also occurs with all subsets of X, with exactly the same support
  - Only reporting A U X is sufficient
Extensions

• Non derivable Itemset Mining [Calders et al, 2002]
  - support bounds of an itemset can be derived from its subsets using the inclusion-exclusion principle
  - if these bounds are tight, then the support of that itemset is derivable
  - only reporting the non-derivable itemsets is sufficient
Outline

• Mining association rules
• Algorithms
  - Apriori
  - Eclat
  - FP-growth
• Optimizations and Extensions
• Other pattern types
• General levelwise search
• Other interestingness measures
Complex Patterns

- Sets
- Sequences
- Graphs
- Relational Structures

- Generation and Counting of such patterns becomes much more complex too!
Sequences

- CGATGGGCCAGTCGATACGTCGATGCCGATGTCACGA
Patterns in Sequences

- Substrings
- Regular expressions \((bb|[^b]{2})\)
- Partial orders
- Directed Acyclic Graphs
- Episodes
Episode mining

• Given a sequence of events
  ABCDBABDABDBSBDADBACBSBACBSBACA
• A sequential episode is an ordered list of events
• Goal: Find all frequently occurring (sequential) episodes
Event sequence: sequence of pairs \((e,t)\), \(e\) is an event, \(t\) an integer indicating the time of occurrence of \(e\).

An linear episode is a sequence of events \(<e_1, ..., e_n>\).

A window of length \(w\) is an interval \([s,e]\) with \((e-s+1) = w\).

An episode \(E=<e_1, ..., e_n>\) occurs in sequence \(S=< (s_1,t_1), ..., (s_m,t_m)>\) within window \(W=[s,e]\) if there exist integers \(s \leq i_1 < ... < i_n \leq e\) such that for all \(j=1...n\), \((e_j,i_j)\) is in \(S\).
The $w$-support of an episode $E = \langle e_1, \ldots, e_n \rangle$ in a sequence $S = \langle (s_1, t_1), \ldots, (s_m, t_m) \rangle$ is the number of windows $W$ of length $w$ such that $E$ occurs in $S$ within window $W$.

Note: If an episode occurs in a very short time span, it will be in many subsequent windows, and thus contribute a lot to the support count!

An episode $E_1 = \langle e_1, \ldots, e_n \rangle$ is a sub-episode of $E_2 = \langle f_1, \ldots, f_m \rangle$, denoted $E_1 \leq E_2$ if there exist integers $1 \leq i_1 < \ldots < i_n \leq m$ such that for all $j = 1 \ldots n$, $e_j = f_{i_j}$. 
Example

- \( S = \langle(b,1), (a,2), (a,3), (c,4), (b,5), (a,6), (a,7), (b,8), (c,9) \rangle \)
- \( E = \langle b, a, c \rangle \)
- \( E \) occurs in \( S \) within window \([0,4]\), within \([1,4]\), within \([5,9]\), ...
- The 5-support of \( E \) in \( S \) is 3, since \( E \) is only in the following windows of length 5: \([0,4]\), \([1,5]\), \([5,9]\)
- \( \langle b, a, a, c \rangle \) is a sub-episode of \( \langle a, b, c, a, a, b, c \rangle \).
Given a sequence w, a minimal support minsup, and a window width w, find all episodes that have a w-support above the minimum support.

Monotonicity
Let S be a sequence, E₁, E₂ episodes, w an integer. If E₁ \leq E₂, then the w-freq(E₂) \leq w-freq(E₁).

We can again apply a level-wise algorithm like Apriori.

Start with small episodes, only proceed with a larger episode if all sub-episodes are frequent.

\(<a,a,b>\) is evaluated after \(<a>, <b>, <a,a>, <a,b>,\) and only if all these episodes were frequent.
Graphs
Patterns and Rules over Graphs

- Graph with f: 5, connected with weight 0.8
- Graph with f: 8, connected with weight 0.5
- Graph with f: 7, connected with weight 0.57
- Graph with f: 4
Relational Databases

**Likes**

<table>
<thead>
<tr>
<th>Drinker</th>
<th>Beer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allen</td>
<td>Duvel</td>
</tr>
<tr>
<td>Allen</td>
<td>Trappist</td>
</tr>
<tr>
<td>Carol</td>
<td>Duvel</td>
</tr>
<tr>
<td>Bill</td>
<td>Duvel</td>
</tr>
<tr>
<td>Bill</td>
<td>Trappist</td>
</tr>
<tr>
<td>Bill</td>
<td>Jupiler</td>
</tr>
</tbody>
</table>

**Visits**

<table>
<thead>
<tr>
<th>Drinker</th>
<th>Bar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allen</td>
<td>Cheers</td>
</tr>
<tr>
<td>Allen</td>
<td>California</td>
</tr>
<tr>
<td>Carol</td>
<td>Cheers</td>
</tr>
<tr>
<td>Carol</td>
<td>California</td>
</tr>
<tr>
<td>Carol</td>
<td>Old Dutch</td>
</tr>
<tr>
<td>Bill</td>
<td>Cheers</td>
</tr>
</tbody>
</table>

**Serves**

<table>
<thead>
<tr>
<th>Bar</th>
<th>Beer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cheers</td>
<td>Duvel</td>
</tr>
<tr>
<td>Cheers</td>
<td>Trappist</td>
</tr>
<tr>
<td>Cheers</td>
<td>Jupiler</td>
</tr>
<tr>
<td>California</td>
<td>Duvel</td>
</tr>
<tr>
<td>California</td>
<td>Jupiler</td>
</tr>
<tr>
<td>Old Dutch</td>
<td>Trappist</td>
</tr>
</tbody>
</table>
Patterns in RDBs

• Query 1:
  - Select L.drinker, V.bar
    From Likes L, Visits V
    Where V.drinker = L.drinker
    And L.beer = 'Duvel'

• Query 2:
  - Select L.drinker, V.bar
    From Likes L, Visits V, Serves S
    Where V.drinker = L.drinker
    And L.beer = 'Duvel'
    And S.bar = V.bar
    And S.beer = 'Duvel'
Patterns in RDBs

• Association Rule:

Query 1 => Query 2

• If a person that likes Duvel visits bar, then that bar serves Duvel
Pattern Mining in general

• Given:
  - A database
  - A partially ordered class of patterns
  - An interestingness measure (e.g. support) which is monotone w.r.t. partial order

• Problem:
  - Find all interesting patterns
Solution

• Generate ‘small’ set of candidate patterns
• Test interestingness measure
• Remove all uninteresting patterns from search space according to monotonicity
• Repeat until all interesting patterns have been found

• [Mannila et al., DMKD 1(3), 1997]
Other constraints or interestingness

• When monotone, Apriori technique can be used
• What if they are not monotone?
• For example:
  - minimum size of itemset or total price of itemset
  - database can be reduced!
• Another example:
  - Mining Tiles
Motivation

• What makes my database unique?

• Describe my database using only a small description

• For example: using itemsets
Motivation

• Which itemsets describe my database best?
• Interestingness measures?
• Most are subjective depending on the specific application
• Support/Frequency is objective
Tiles

• A *tile* is an itemset together with the transactions in which it occurs
• We only consider maximal tiles (= closed)
Tile Mining

• The area of a tile is the number of 1’s occurring in it
• Goal: Find all tiles with area at least s
Can we efficiently find them?

- Area of tiles is not monotone w.r.t. set inclusion 😞

- Mining tiles and tilings is NP-hard 😞 (~maximum edge biclique problem)
The LTM algorithm

- Branch and bound
- Traverse itemset lattice depth-first (like Eclat and FP-growth)
- At every node, bound the size of the largest tile that can still be found
The bound

- For every item, we count the number of transactions of size larger than $k$ in which the item occurs.

<table>
<thead>
<tr>
<th></th>
<th>100</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
<td>160</td>
</tr>
<tr>
<td>3</td>
<td>60</td>
<td>180</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>160</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
The Dynamics

- If an item can not occur in a large tile anymore, we can remove it.
- If a transaction can not contribute to a large tile anymore, we can remove it.
- If an item in a specific transaction can not contribute to a large tile, we can remove it from that transaction.
- Results in shorter transactions.
- Recompute the bounds.
The End

C++ Implementations of Apriori, Eclat, FP-growth and several other algorithms are available on my webpage http://www.adrem.ua.ac.be/~goethals/software/ and on http://fimi.cs.helsinki.fi/

Sources: I used some material from slides of Jiawei Han and Toon Calders