Data mining

Piotr Paszek

Association Rule Mining
Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction.

- Market-Basket transactions

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bread, Milk</td>
</tr>
<tr>
<td>2</td>
<td>Bread, Diaper, Beer, Eggs</td>
</tr>
<tr>
<td>3</td>
<td>Milk, Diaper, Beer, Coke</td>
</tr>
<tr>
<td>4</td>
<td>Bread, Milk, Diaper, Beer</td>
</tr>
<tr>
<td>5</td>
<td>Bread, Milk, Diaper, Coke</td>
</tr>
</tbody>
</table>

Example of Association Rules

\[
\{\text{Diaper}\} \rightarrow \{\text{Beer}\}, \quad \{\text{Beer, Bread}\} \rightarrow \{\text{Milk}\}\]
Association Rule Definitions

- $I = i_1, i_2, \ldots, i_n$: a set of all the items
- Transaction $T$: a set of items such that $T \subset I$
- Transaction Database $D$: a set of transactions
- A transaction $T$ contains a set $X \subset I$ of some items, if $X \subset T$

**An Association Rule**: is an implication of the form

$$X \rightarrow Y,$$

where $X, Y \subset I$. 
A set of items is referred as an *itemset*. A itemset that contains $k$ items is a $k$-itemset.

The *support* ($sup$) of an itemset $X$ is the percentage of transactions in the transaction database $D$ that contain $X$.

*Frequent Itemset*: An itemset whose support is greater than or equal to a $minsup$ threshold.
Why do we want to find frequent itemsets?

Motivation: Finding inherent regularities in data
- What products were often purchased together? – Beer and diapers?!
- What kinds of DNA are sensitive to this new drug?
- Can we automatically classify web documents?

Applications
- Basket data analysis, cross-marketing, catalog design, sale campaign analysis, Web log (click stream) analysis, and DNA sequence analysis.
The **support** \((sup)\) of the rule \(X \rightarrow Y\) in the transaction database \(D\) is the support of the items set \(X \cup Y\) in \(D\) (fraction of transactions that contain both \(X\) and \(Y\)).

\[
sup(X \rightarrow Y) = \frac{|\{T \in D : X \subseteq T \land Y \subseteq T\}|}{|D|}
\]

The **confidence** \((conf)\) of the rule \(X \rightarrow Y\) in the transaction database \(D\) is the ratio of the number of transactions in \(D\) that contain \(X \cup Y\) to the number of transactions that contain \(X\) in \(D\) (measures how often items in \(Y\) appear in transactions that contain \(X\)).

\[
conf(X \rightarrow Y) = \frac{|\{T \in D : X \subseteq T \land Y \subseteq T\}|}{|\{T \in D : X \subseteq T\}|}
\]
Given:

- a set \( I \) of all the items;
- a database \( D \) of transactions;
- minimum support: \( \text{minsup} \);
- minimum confidence: \( \text{minconf} \).

Find:

- all association rules \( X \rightarrow Y \) having support greater than or equal to a \( \text{minsup} \) threshold \( (\text{sup} \geq \text{minsup}) \) and confidence greater than or equal to a \( \text{minconf} \) \( (\text{conf} \geq \text{minconf}) \)
Finding Association Rule – Brute-force approach

Brute-force approach

1. List all possible association rules
2. Compute the support and confidence for each rule
3. Prune rules that fail the \textit{minsup} and \textit{minconf} thresholds

Computationally prohibitive!
Computational complexity of step 1:

- \# of possible association rules $= 2^{|I|} \cdot 2^{|I|} = 2^{2|I|}$
- $I$ – a set of items
Association Rule Mining Task

Problem Decomposition

Two-step approach:

- **Frequent Itemset Generation**
  Generate all itemsets whose \( support \geq minsup \) (frequent itemsets)

- **Rule Generation**
  Generate high confidence rules from each frequent itemset, where each rule is a binary partitioning of a frequent itemset

- Frequent itemset generation is still computationally expensive
Frequent Itemset Generation

Given $|I|$ items, there are $2^{|I|}$ possible itemsets (power set $I$). 

(null)

A

B

C

D

E

AB

AC

AD

AE

BC

BD

BE

CD

CE

DE

ABC

ABD

ABE

ACD

ACE

ADE

BCD

BCE

BDE

CDE

ABCD

ABCE

ABDE

ACDE

BCDE

ABCDE
Frequent Itemset Generation

Brute-force approach:

- Each itemset (from $I$) is a candidate frequent itemset
- Count the support of each candidate by scanning the database
- Match each transaction against every candidate

Computational complexity:

$\approx O(|D| \cdot 2^{|I|})$
Frequent Itemset Generation Strategies

- Reduce the number of candidates ($M$)
  - Complete search: $M = 2^d$
  - Use pruning techniques to reduce $M$
- Reduce the number of transactions ($N$)
  - Reduce size of $N$ as the size of itemset increases
  - Used vertical-based mining algorithms
- Reduce the number of comparisons ($NM$)
  - Use efficient data structures to store the candidates or transactions
  - No need to match every candidate against every transaction
There Are Too Many Frequent Itemsets!

- A long itemset (pattern) contains a combinatorial number of sub-patterns
- How to deal with such a problem?
- Expressing Itemsets in Compressed Form:
  - **Closed itemsets**
    An itemset is closed if is frequent and none of its immediate supersets has the same support as the itemset
    Mining Frequent Closed Itemsets: CLOSET
  - **Maximal Itemset**
    An itemset is maximal frequent if is frequent and none of its immediate supersets is frequent
    Mining Maximal Itemsets: MaxMiner
Maximal vs Closed Itemsets

- \( DB = \{ < a_1, \ldots, a_{100} >, < a_1, \ldots, a_{50} > \} \)
- \( \text{Min}sup = 1. \)

What is the set of closed itemset?
- \( < a_1, \ldots, a_{100} > \)
- \( < a_1, \ldots, a_{50} > \)

What is the set of maximal itemset?
- \( < a_1, \ldots, a_{100} > \)

What is the set of all patterns?
- ?!
Maximal vs Closed Itemsets
Reducing Number of Candidates

- **Apriori principle** (Frequent Itemset Property):
  - If an itemset is frequent, then all of its subsets must also be frequent
- **Contrapositive:**
  - If an itemset is not frequent, none of its supersets are frequent.
Apriori principle
Apriori principle (Contrapositive)

Found to be Infrequent

Pruned supersets
Scalable Methods for Mining Frequent Itemset

- Apriori Algorithm (Agrawal & Srikant 1994)
- Frequent pattern growth Algorithm (FP–growth) (Han, Pei & Yin 2000)
- Vertical data format approach (Charm) (Zaki & Hsiao 2002)
Apriori Algorithm

Method:

- Initially, scan DB once to get frequent 1-itemset \((k = 1)\)
- Repeat until no new frequent itemsets are identified
  - Generate length \((k + 1)\) candidate itemsets from length \(k\) frequent itemsets
  - Prune candidate itemsets containing subsets of length \(k\) that are infrequent
  - Count the support of each candidate by scanning the DB
  - Eliminate candidates that are infrequent, leaving only those that are frequent
Apriori Algorithm

Let:

- $L_k$ – set of frequent itemsets of size $k$ (with min support)
- $C_k$ – set of candidate itemsets of size $k$
  (potentially frequent itemsets)
- $\text{apriori\_gen}()$ – a function that generates candidate itemsets
Apriori Algorithm – Pseudo-code

Apriori$(minsup)$

$L_1 \leftarrow \{\text{frequent items}\}$;
for (k=2; $L_{k-1} \neq \emptyset$; k++) do
begin
$C_k \leftarrow \text{apriori-gen}(L_{k-1})$;
forall $t \in T$ do
begin
$C_t \leftarrow \text{subset}(C_k, t)$;
forall $c \in C_t$ do $c$.count++;
end;
$L_k \leftarrow \{c \in C_k | c$.count $\geq minsup\}$
end;
Return $(\bigcup_k L_k)$;
function apriori_gen($C_k$)
    insert into $C_k$       //self-joining
    select $p.item_1, p.item_2, \ldots, p.item_{k-1}, q.item_{k-1}$
    from $p, q \in L_{k-1}$
    where $p.item_1 = q.item_1, \ldots, p.item_{k-2} = q.item_{k-2},$
    $p.item_{k-1} < q.item_{k-1};$
    forall $c \in C_k$ do
        forall $(k-1)$-subsets $s$ of $c$ do
            if $(s \not\in L_{k-1})$ then    //pruning
                delete $c$ from $C_k;$
        endfor;
    endfor;
endfunction;
Apriori Algorithm

Computational complexity:

- \( \text{apriori\_gen()} \approx O(|L_k|^3) \)
- \( \text{Apriori()} \approx O(\sum_k \{|L_k|^3 + |C_k||D||I|\}) \).
- \text{Apriori} algorithm requires \( k + 1 \) database scans;
  \( k \) is the maximum size of frequent itemset
Factors Affecting Complexity

- Choice of minimum support threshold
  - lowering support threshold results in more frequent itemsets
  - this may increase number of candidates and max length of frequent itemsets

- Dimensionality (number of items) of the data set
  - more space is needed to store support count of each item
  - if number of frequent items also increases, both computation and I/O costs may also increase

- Size of database
  - since \textit{Apriori} makes multiple passes, run time of algorithm may increase with number of transactions

- Average transaction width
  - transaction width increases with denser data sets
  - this may increase maximal length of frequent itemsets
The Apriori Algorithm – Example

Database D

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1 3 4</td>
</tr>
<tr>
<td>200</td>
<td>2 3 5</td>
</tr>
<tr>
<td>300</td>
<td>1 2 3 5</td>
</tr>
<tr>
<td>400</td>
<td>2 5</td>
</tr>
</tbody>
</table>

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>

Min support = 50%

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>

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

C₂

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

L₂

C₃

<table>
<thead>
<tr>
<th>itemset</th>
<th>sup.</th>
</tr>
</thead>
<tbody>
<tr>
<td>{2 3 5}</td>
<td></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>
### Advantages:
- Uses large itemset property.
- Easily parallelized
- Easy to implement.

### Disadvantages:
- Assumes transaction database is memory resident.
- Requires many database scans.
Generate high confidence rules from each frequent itemset, where each rule is a binary partitioning of a frequent itemset

**Pseudo-code**

```
for each frequent itemset $F$ do
    for each subset $C$ of $F$ do
        if $(\text{supp}(F)/\text{supp}(F - C) \geq \text{minconf})$ then
            output the rule $(F - C) \rightarrow C$
            with $\text{conf} = \text{supp}(F)/\text{supp}(F - C)$
            and $\text{supp} = \text{supp}(F)$
        endif
    endfor
endfor
```
Example of Rules Generation

<table>
<thead>
<tr>
<th>TID</th>
<th>List of Item_IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>T100</td>
<td>I1, I2, I5</td>
</tr>
<tr>
<td>T200</td>
<td>I2, I4</td>
</tr>
<tr>
<td>T300</td>
<td>I2, I3</td>
</tr>
<tr>
<td>T400</td>
<td>I1, I2, I4</td>
</tr>
<tr>
<td>T500</td>
<td>I1, I3</td>
</tr>
<tr>
<td>T600</td>
<td>I2, I3</td>
</tr>
<tr>
<td>T700</td>
<td>I1, I3</td>
</tr>
<tr>
<td>T800</td>
<td>I1, I2, I3, I5</td>
</tr>
<tr>
<td>T900</td>
<td>I1, I2, I3</td>
</tr>
</tbody>
</table>

Let us consider the 3-itemset \{I1, I2, I5\} with support of 0.22(2)%. Let generate all the association rules from this itemset:

- I1 $\land$ I2 $\Rightarrow$ I5 confidence = $2/4 = 50\%$
- I1 $\land$ I5 $\Rightarrow$ I2 confidence = $2/2 = 100\%$
- I2 $\land$ I5 $\Rightarrow$ I1 confidence = $2/2 = 100\%$
- I1 $\Rightarrow$ I2 $\land$ I5 confidence = $2/6 = 33\%$
- I2 $\Rightarrow$ I1 $\land$ I5 confidence = $2/7 = 29\%$
- I5 $\Rightarrow$ I1 $\land$ I2 confidence = $2/2 = 100\%$
Frequent Pattern Growth Algorithm

- **FP-Growth:** allows frequent itemsets discovery without candidate itemsets generation.

**Two step approach:**

- **Step 1:** Build a compact data structure called the FP-tree. Built using 2 passes over the data-set.
- **Step 2:** Extracts frequent itemsets directly from the FP-tree.
FP-Growth Algorithm

Step 1: FP-Tree Construction

FP-Tree is constructed using 2 passes over the data-set:
Pass 1:
- Scan data and find support for each item.
- Discard infrequent items.
- Sort frequent items in decreasing order based on their support.

Use this order when building the FP-Tree, so common prefixes can be shared.
FP-Growth Algorithm

Step 1: FP-Tree Construction

Pass 2:
Nodes correspond to items and have a counter

- FP-Growth reads 1 transaction at a time and maps it to a path
- Fixed order is used, so paths can overlap when transactions share items (when they have the same prefix).
  - In this case, counters are incremented
- Pointers are maintained between nodes containing the same item, creating singly linked lists (dotted lines)
  - The more paths that overlap, the higher the compression.
FP-tree may fit in memory.
- Frequent itemsets extracted from the FP-Tree.
The FP-Tree usually has a smaller size than the uncompressed data - typically many transactions share items (and hence prefixes).

- Best case scenario: all transactions contain the same set of items.
  - 1 path in the FP-tree
- Worst case scenario: every transaction has a unique set of items (no items in common)
  - Size of the FP-tree is at least as large as the original data.
  - Storage requirements for the FP-tree are higher - need to store the pointers between the nodes and the counters.

The size of the FP-tree depends on how the items are ordered.

Ordering by decreasing support is typically used but it does not always lead to the smallest tree (it’s a heuristic).
FP-Growth Algorithm

Step 2: Frequent Itemset Generation

- FP-Growth extracts frequent itemsets from the FP-tree.
- Bottom-up algorithm – from the leaves towards the root.
- Divide and conquer:
  first look for frequent itemsets ending in $e$, then $de$, etc . . . then $d$, then $cd$, etc . . .
- First, extract prefix path sub-trees ending in an item(set).
  (hint: use the linked lists)
### FP-Growth Advantages/Disadvantages

#### Advantages:
- only 2 passes over data-set
- “compresses” data-set
- no candidate generation
- much faster than Apriori

#### Disadvantages:
- FP-Tree may not fit in memory!!
- FP-Tree is expensive to build