Frequent patterns

Humpty Dumpty sat on a wall,
Humpty Dumpty had a great fall;
All the King's horses and all the King's men
Couldn't put Humpty together again

Frequent patterns

Humpty Dumpty sat on a wall,
Humpty Dumpty had a great fall;
All the King's horses and all the King's men
Couldn't put Humpty together again

Frequent patterns

Humpty Dumpty sat on a wall,
Humpty Dumpty had a great fall;
All the King's horses and all the King's men
Couldn't put Humpty together again

Frequent patterns

Humpty Dumpty sat on a wall,
Humpty Dumpty had a great fall;
All the King's horses and all the King's men
Couldn't put Humpty together again

<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>
Frequent patterns

Why frequent patterns

Quiz
Frequent itemsets

› In general, we may be interested in different types of data and types of patterns:

› Natural language / Word sequences
› Text / Regular expression patterns
› Web logs / Event sequences (various models)
› Purchases / Item sets

Today

Let the data consist of transactions (i.e. sets of items)

<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>

› The patterns we are interested are itemsets.
› E.g. \{Milk, Eggs\} is an itemset.

Definitions

The support of an itemset is the proportion of transactions where it occurs

<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>

› support({Milk, Eggs}) = 0
› support({Bread}) = 4/5 = 0.8
› support({Milk, Beer}) = 2/5 = 0.4

Definitions

The support of an itemset is the proportion of transactions where it occurs

<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>

› support({Milk, Eggs}) = 0
› support({Bread}) = 4
› support({Milk, Beer}) = 2

Explanations for itemsets

› Suppose we found that \{Beer, Milk, Diapers\} is a frequent itemset.
› How do we explain it?

Perhaps it is frequent due to the fact that there is a causal relationship
\{Diapers, Milk\} => Beer
Association rules

- An association rule is an implication of the form $X \rightarrow Y$, where $X$ and $Y$ are itemsets.
- The support of a rule is just the support of $X \cup Y$.
  $$\text{support}(X \rightarrow Y) = \text{support}(X \cup Y)$$
- The confidence of a rule is the proportion of transactions satisfying the right part among the transactions, which satisfy the left.
  $$\text{confidence}(X \rightarrow Y) = \frac{\text{support}(X \cup Y)}{\text{support}(X)}$$

Frequent itemset mining

- Brute force approach:
  - Enumerate all possible sets of items
  - For each set compute its support in the database
  - Output sets with support $\geq s_{\text{min}}$

Association rule mining

Given a set of transactions, find all rules $X \rightarrow Y$, such that

- $\text{support}(X \rightarrow Y) \geq s_{\text{min}}$
- $\text{confidence}(X \rightarrow Y) \geq c_{\text{min}}$

Solution:

- Find frequent itemset $A$ with support $\geq s_{\text{min}}$
- Then find a partitioning of $A$ into left- and right-part, so that the resulting rule has high confidence.
- The algorithmic approach is the same in both steps. We'll start with the first one.

Faster itemset mining

- Apriori: Avoid scanning through all $2^d$ itemsets.
- FP-tree: Also store transactions in an efficient data structure, speeding up matching.
The Apriori idea

- Suppose that
  \( \text{support}(\{A, C\}) = 42/100 \)
- It follows that
  \( \text{support}(\{A, B, C\}) \leq 42/100 \)

Anti-monotonicity of support

In general,

\[ X \subseteq Y \Rightarrow \text{support}(X) \geq \text{support}(Y) \]

it follows that:

- If an itemset is not frequent, all of its supersets are also not frequent.
- and
  - If an itemset is frequent, all of its subsets are also frequent.

Apriori principle

Found to be infrequent

Pruned supersets

Basic Apriori algorithm

- First generate frequent 1-sets,
- Next, generate frequent 2-sets from 1-sets,
  - ... then generate frequent 3-sets from 2-sets,
  - ... etc, until there are no frequent \( k \)-sets

Basic Apriori algorithm

- Next, generate frequent 2-sets. This is done in several steps.

1. Generate candidate 2-sets

| \{A\} | 10 |
| \{B\} | 15 |
| \{C\} | 5  |
| \{E\} | 6  |
| \{F\} | 10 |

Let min support count = 5

All subsets of a candidate set must be frequent.
For 2-sets it simply means that both elements are from frequent 1-sets.
Basic Apriori algorithm

Next, generate frequent 2-sets. This is done in several steps.

1. Generate candidate 2-sets
   - \( \{A\} \): 10
   - \( \{B\} \): 15
   - \( \{E\} \): 6
   - \( \{F\} \): 10
   - \( \{A, B\} \): 10
   - \( \{A, F\} \): 5
   - \( \{B, E\} \): 6
   - \( \{B, F\} \): 10
   - \( \{E, F\} \): 5

   (requires a full pass over the transaction database for each candidate)

2. Count actual support for each candidate
   - \( \{A\} \): 10
   - \( \{B\} \): 15
   - \( \{E\} \): 6
   - \( \{F\} \): 10
   - \( \{A, B\} \): 10
   - \( \{A, F\} \): 5
   - \( \{B, E\} \): 6
   - \( \{B, F\} \): 10
   - \( \{E, F\} \): 5

   (Again, a pass over the whole DB)

3. ... and leave only actually frequent ones

Now we have frequent 1- and 2-sets.

In many practical situations the algorithm stops here.

1. Because there are so many items that enumerating beyond 2-sets is impractical.
2. Because knowing frequent 2-sets is already useful enough (think of the "Beer/Dispers" example)

But let's try generating frequent 3-sets. We proceed as before.

1. Generate candidate 3-sets
   - \( \{A\} \): 10
   - \( \{B\} \): 15
   - \( \{E\} \): 6
   - \( \{F\} \): 10
   - \( \{A, B\} \): 10
   - \( \{A, F\} \): 5
   - \( \{B, E\} \): 6
   - \( \{B, F\} \): 10
   - \( \{E, F\} \): 5

   We augment each 2-set with an additional element and check that all 2-subsets of the resulting set is frequent.

But let's try generating frequent 3-sets. We proceed as before.

1. Generate candidate 3-sets
   - \( \{A\} \): 10
   - \( \{B\} \): 15
   - \( \{E\} \): 6
   - \( \{F\} \): 10
   - \( \{A, B\} \): 10
   - \( \{A, F\} \): 5
   - \( \{B, E\} \): 6
   - \( \{B, F\} \): 10
   - \( \{E, F\} \): 5

   (Again, a pass over the whole DB)

3. ... and throw away the non-frequent ones

This can be optimized somewhat, see, e.g. http://www.dais.unive.it/~orlando/PAPERS/dawak01.pdf
Basic Apriori algorithm

Quiz: In this particular example, can there be frequent 4-sets?

No, because if, say, \( \{B, E, F, X\} \) is frequent, then \( \{B, E, X\} \) must be frequent too!

Many optimizations are possible

- Avoid generating the separate candidate set explicitly (do it on-the-fly while counting).
- Store \( k \)-sets in a hash tree data structure, (speeds up the counting/generation process).
- Use a part of the whole transaction DB (sample or partition)
- Use Bloom-filter like data structures to reduce candidate set.
- etc.

Compact representation of itemsets

If \( \{A, B, C, D, E\} \) is frequent, then also those sets are frequent:

\[
\begin{align*}
&\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \\
&\{A, B\}, \{A, C\}, \{A, D\}, \{A, E\}, \{B, C\}, \{B, D\}, \{B, E\}, \{C, D\}, \{C, E\}, \{D, E\}, \{A, B, C\}, \{A, B, D\}, \{A, B, E\}, \{A, C, D\}, \{A, C, E\}, \{A, D, E\}, \{B, C, D\}, \{B, C, E\}, \{B, D, E\}, \{C, D, E\}
\end{align*}
\]

Do we really want our algorithm to report those?
Quiz

- Can an itemset be:
  - Closed and not Maximal? Yes.
  - E.g., adding any item will reduce support, but adding some items will still make a frequent set.
  - Not closed and Maximal? No.
  - Not closed, hence we can add some other item without reducing support.
  - Closed and Maximal? Yes.
  - Adding any item will reduce support and make the set infrequent.

Intermediate summary

- **Frequent itemsets** are interesting because those correspond to **structure in the data**.
- **Association rules** are basically frequent itemsets with bells and whistles.
- **Apriori-like algorithms** search for frequent sets better than brute-force.
- It is sufficient to find only **maximal itemsets** (or **closed itemsets**, if some flexibility in changing cutoff later is needed).

Apriori is fine, but

- When the number of frequent patterns is large, the candidate sets are large and the repeated database scans are slow.
- This happens, in particular, when there is a long frequent pattern (then all subsets are also frequent).

FP-tree

- A frequent-pattern tree is a data structure for storing the transaction database, which allows to enumerate frequent itemsets.

Constructing the FP-tree

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
<th>1. Count the frequency of each item</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>A,C,D,G,I,M,P</td>
<td>Frequent items</td>
</tr>
<tr>
<td>300</td>
<td>B,F,H,J,O</td>
<td>A: 3  B: 3  C: 4  L: 2  M: 2  N: 3  O: 2  P: 1  R: 1</td>
</tr>
</tbody>
</table>
Constructing the FP-tree

2. Leave only the frequent items

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>F,C,A,M,P</td>
</tr>
<tr>
<td>200</td>
<td>A,B,C,F,M</td>
</tr>
<tr>
<td>300</td>
<td>B,F</td>
</tr>
<tr>
<td>400</td>
<td>B,C,P</td>
</tr>
<tr>
<td>500</td>
<td>A,F,C,RM</td>
</tr>
</tbody>
</table>

F: 4  
A: 3  
C: 4  
B: 3  
M: 3  
P: 3

Constructing the FP-tree

3. Sort items by frequency (recommended)

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>F,C,A,M,P</td>
</tr>
<tr>
<td>200</td>
<td>F,C,A,B,M</td>
</tr>
<tr>
<td>300</td>
<td>F,B</td>
</tr>
<tr>
<td>400</td>
<td>C,B,P</td>
</tr>
<tr>
<td>500</td>
<td>F,C,A,M,P</td>
</tr>
</tbody>
</table>

F: 4  
C: 4  
A: 3  
B: 3  
M: 3  
P: 3

Constructing the FP-tree

4. Create a header table and a root node

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>F,C,A,M,P</td>
</tr>
<tr>
<td>200</td>
<td>F,C,A,B,M</td>
</tr>
<tr>
<td>300</td>
<td>F,B</td>
</tr>
<tr>
<td>400</td>
<td>C,B,P</td>
</tr>
<tr>
<td>500</td>
<td>F,C,A,M,P</td>
</tr>
</tbody>
</table>

Constructing the FP-tree

5. Now start inserting transaction. Each transaction will be a path in the tree.

Next transaction shares a part of the path (F,C,A) with the previously inserted. We keep counts at each node to track this.
Constructing the FP-tree

We update header table lists (the new nodes B and M need to be included)

Constructing the FP-tree

Inserting third transaction...

... fourth

... fifth is exactly like the first, so we only update counts along the path

The resulting data structure has all the information from the original database, but allows some fast lookups.

Quiz

How to find out the total number of transactions stored in this tree?
Quiz

How many transactions contain “M”?

Frequent-pattern enumeration based on FP-tree

Core idea:
- Recursively enumerate all subsets.
- For each subset construct a FP-tree of transactions that contain this subset.
- Output those subsets for which the corresponding tree contains more than (support count) transactions.
FP-growth algorithm

Frequent-pattern enumeration based on FP-tree

Core idea:
- Recursively enumerate all subsets.

```python
def subsets(items, current_subset):
    for i in items:
        if i > max(current_subset):
            new_subset = current_subset + [i]
            subset(items, new_subset)
```

Start enumeration with `subsets(items, [])`

20.02.2014 MTAT.03.183 Data Mining - Frequent Itemsets

FP-growth algorithm

Frequent-pattern enumeration based on FP-tree

Core idea:
- Recursively enumerate all subsets.

```python
def subsets(items, current_subset, FP_tree):
    process(current_subset)
    for i in items:
        if i > max(current_subset):
            new_subset = current_subset + [i]
            new_tree = FP_tree.filter(i)
            subsets(items, new_subset, new_tree)
```

Start enumeration with `subsets(items, [], full_tree)`

20.02.2014 MTAT.03.183 Data Mining - Frequent Itemsets

FP-growth algorithm

Frequent-pattern enumeration based on FP-tree

Start from the bottom of the header table

Extract all transactions that contain "P" and make a corresponding FP-tree

20.02.2014 MTAT.03.183 Data Mining - Frequent Itemsets

FP-growth algorithm

Frequent-pattern enumeration based on FP-tree

Start from the bottom of the header table

Extract all transactions that contain "P" and make a corresponding FP-tree

20.02.2014 MTAT.03.183 Data Mining - Frequent Itemsets

FP-growth algorithm

Frequent-pattern enumeration based on FP-tree

We can see that P is frequent (for support threshold 3).

Now descend recursively to examine all sets like P+{...}, start with (P,M)

20.02.2014 MTAT.03.183 Data Mining - Frequent Itemsets

FP-growth algorithm

Frequent-pattern enumeration based on FP-tree

We can see that P is frequent (for support threshold 3).

Now descend recursively to examine all sets like P+{...}, start with (P,M)
FP-growth algorithm

Frequent-pattern enumeration based on FP-tree

It is not frequent, so no need to dig deeper with \{P,M, \ldots\}

We can return from the recursive call

Intermediate summary

- **Frequent itemsets** are interesting because those correspond to **structure in the data**.
- **Association rules** are basically frequent itemsets with bells and whistles.
- **Apriori-like algorithms** search for frequent sets better than brute-force.
- **FP-tree** is a data structure for keeping transactions.
- **FP-growth** can be more memory-efficient than Apriori.
- **Maximal** and **closed** itemsets are your friends.
Back to association rules

Recall association rule mining:

- Problem: find association rules \( X \rightarrow Y \) with
  - Support at least \( s_{\text{min}} \)
  - Confidence at least \( c_{\text{min}} \)

- Solution:
  - Find frequent itemsets with support at least \( s_{\text{min}} \)
  - For each itemset find a split into \( X \rightarrow Y \), which ensures required confidence.

Rule generation

- Recall that
  \[
  \text{confidence}(A \rightarrow B, C, D) = \frac{\text{support}(A, B, C, D)}{\text{support}(A)}
  \]

  \[
  \text{confidence}(A, B \rightarrow C, D) = \frac{\text{support}(A, B, C, D)}{\text{support}(A, B)}
  \]

  \[
  \text{confidence}(A, C, D \rightarrow B) = \frac{\text{support}(A, B, C, D)}{\text{support}(A, C, D)}
  \]

Rule generation

- Suppose we found that \( \{A, B, C, D\} \) is a frequent itemset with the necessary support.

- We can make a variety of rules from it:
  - \( \{A\} \rightarrow \{B, C, D\} \)
  - \( \{A, B\} \rightarrow \{C, D\} \)
  - \( \{B, C, D\} \rightarrow \{A\} \)

- How to efficiently find those which satisfy the confidence threshold?

Rule generation

- Recall that
  \[
  \text{confidence}(A \rightarrow B, C, D) = \frac{\text{support}(A, B, C, D)}{\text{support}(A)}
  \]

- Consequently, among all rules built on the set \( \{A, B, C, D\} \), confidence is inverse proportional to the support of antecedent (the left part of the rule).

- I.e. confidence is monotonic wrt antecedent (and anti-monotonic wrt consequent).

Quiz

- Could you use FP-tree with the “usual” Apriori algorithm?

- Can you apply FP-growth to search for the rule split for a given frequent set?