Frequent Itemset Mining and The Apriori algorithm

Philippe Fournier-Viger
http://www.philippe-Fournier-viger.com


Source code and datasets available in the SPMF library
Introduction

- Many retail stores collect data about customers.
- e.g. *customer transactions*
- Need to analyze this data to understand customer behavior
- Why?
  - for marketing purposes,
  - inventory management,
  - customer relationship management
Introduction

Discovering patterns and associations

• Discovering interesting relationships hidden in large databases.
• e.g. beer and diapers are often sold together
• pattern mining is a fundamental data mining problem with many applications in various fields.
• Introduced by Agrawal (1993).
• Many extensions of this problem to discover patterns in graphs, sequences, and other kinds of data.
FREQUENT ITEMSET MINING
Definitions

Let $I = \{I_1, I_2, \ldots, I_m\}$ be the set of items (products) sold in a retail store.

For example:

$I = \{\text{pasta, lemon, bread, orange, cake}\}$
Definitions

A **transaction database** $D$ is a set of **transactions**. $D = \{T_1, T_2, \ldots, T_r\}$ such that $T_a \subseteq I \ (1 \leq a \leq r)$.

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items appearing in the transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>{pasta, lemon, bread, orange}</td>
</tr>
<tr>
<td>T2</td>
<td>{pasta, lemon}</td>
</tr>
<tr>
<td>T3</td>
<td>{pasta, orange, cake}</td>
</tr>
<tr>
<td>T4</td>
<td>{pasta, lemon, orange, cake}</td>
</tr>
</tbody>
</table>
Definitions

Each transaction has a unique identifier called its **Transaction ID (TID)**.

*e.g.* the transaction ID of $T_4$ is 4.

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items appearing in the transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>{pasta, lemon, bread, orange}</td>
</tr>
<tr>
<td>T2</td>
<td>{pasta, lemon}</td>
</tr>
<tr>
<td>T3</td>
<td>{pasta, orange, cake}</td>
</tr>
<tr>
<td>T4</td>
<td>{pasta, lemon, orange, cake}</td>
</tr>
</tbody>
</table>
A transaction is a set of items (an itemset).

e.g. \( T_2 = \{\text{pasta, lemon}\} \)

An item (a symbol) may not appear or appear once in each transaction. Each transaction is unordered.

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items appearing in the transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_1 )</td>
<td>{pasta, lemon, bread, orange}</td>
</tr>
<tr>
<td>( T_2 )</td>
<td>{pasta, lemon}</td>
</tr>
<tr>
<td>( T_3 )</td>
<td>{pasta, orange, cake}</td>
</tr>
<tr>
<td>( T_4 )</td>
<td>{pasta, lemon, orange, cake}</td>
</tr>
</tbody>
</table>
A transaction database can be viewed as a binary matrix:

<table>
<thead>
<tr>
<th>Transaction</th>
<th>pasta</th>
<th>lemon</th>
<th>bread</th>
<th>orange</th>
<th>cake</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Asymmetrical binary attributes (because 1 is more important than 0)
- There is no information about purchase quantities and prices.
Definitions

Let \( I \) be the set of all items:

\[ I = \{ \text{pasta, lemon, bread, orange, cake} \} \]

There are \( 2^{|I|} - 1 = 2^5 - 1 = 31 \) subsets:

\{pasta\}, \{lemon\}, \{bread\}, \{orange\}, \{cake\}
\{pasta, lemon\}, \{pasta, bread\} \{pasta, orange\}, \{pasta, cake\}, \{lemon, bread\}, \{lemon, orange\},
\{lemon, cake\}, \{bread, orange\}, \{bread, cake\}

\[ \ldots \]
\{pasta, lemon, bread, orange, cake\}
Definitions

An itemset is said to be of size $k$, if it contains $k$ items.

Itemsets of size 1:

{pasta}, {lemon}, {bread}, {orange}, {cake}

Itemsets of size 2:

{pasta, lemon}, {pasta, bread} {pasta, orange}, {pasta, cake}, {lemon, bread}, {lemon orange}, …
The **support (frequency)** of an itemset \( X \) is the number of transactions that contains \( X \).

\[
\text{sup}(X) = |\{T \mid X \subseteq T \land T \in D\}|
\]

**For example:** The support of \( \{\text{pasta, orange}\} \) is 3.

which is written as: \( \text{sup}(\{\text{pasta, orange}\}) = 3 \)

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items appearing in the transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>{pasta, lemon, bread, orange}</td>
</tr>
<tr>
<td>T2</td>
<td>{pasta, lemon}</td>
</tr>
<tr>
<td>T3</td>
<td>{pasta, orange, cake}</td>
</tr>
<tr>
<td>T4</td>
<td>{pasta, lemon, orange, cake}</td>
</tr>
</tbody>
</table>
Definitions

The support of an itemset $X$ can also be written as a ratio (absolute support).

Example: The support of \{pasta, orange\} is 75% because it appears in 3 out of 4 transactions.

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items appearing in the transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>{pasta, lemon, bread, orange}</td>
</tr>
<tr>
<td>T2</td>
<td>{pasta, lemon}</td>
</tr>
<tr>
<td>T3</td>
<td>{pasta, orange, cake}</td>
</tr>
<tr>
<td>T4</td>
<td>{pasta, lemon, orange, cake}</td>
</tr>
</tbody>
</table>
The problem of frequent itemset mining

- Let there be a numerical value $\text{minsup}$, set by the user.
- Frequent itemset mining (FIM) consists of enumerating all frequent itemsets, that is itemsets having a support greater or equal to $\text{minsup}$.

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items appearing in the transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>{pasta, lemon, bread, orange}</td>
</tr>
<tr>
<td>T2</td>
<td>{pasta, lemon}</td>
</tr>
<tr>
<td>T3</td>
<td>{pasta, orange, cake}</td>
</tr>
<tr>
<td>T4</td>
<td>{pasta, lemon, orange, cake}</td>
</tr>
</tbody>
</table>
For \textit{minsup} = 2, the frequent itemsets are:

\{lemon\}, \{pasta\}, \{orange\}, \{cake\}, \{lemon, pasta\}, \{lemon, orange\}, \{pasta, orange\}, \{pasta, cake\}, \{orange, cake\}, \{lemon, pasta, orange\}

For the user, choosing a high \textit{minsup} value,

- will reduce the number of frequent itemsets,
- will increase the speed and decrease the memory required for finding the frequent itemsets
Numerous applications

Frequent itemset mining has numerous applications.

- medical applications,
- chemistry,
- biology,
- e-learning,
- etc.
Several algorithms

Algorithms:
- **Apriori, AprioriTID** *(1993)*
- Eclat *(1997)*
- **FPGrowth** *(2000)*
- **Hmine** *(2001)*
- **LCM**, ...
- ...

Moreover, numerous extensions of the FIM problem: uncertain data, fuzzy data, purchase quantities, profit, weight, time, rare itemsets, closed itemsets, etc.
ALGORITHMS
Naïve approach

- If there are $n$ items in a database, there are $2^n - 1$ itemsets may be frequent.
- **Naïve approach**: count the support of all these itemsets.
- To do that, we would need to read each transaction in the database to count the support of each itemset.
- This would be inefficient:
  - need to perform too many comparisons
  - requires too much memory
Search space

This is all the itemsets that can be formed with the items lemon (l), pasta (p), bread (b), orange (o) and cake (c)

\[ l = \text{lemon} \]
\[ p = \text{pasta} \]
\[ b = \text{bread} \]
\[ o = \text{orange} \]
\[ c = \text{cake} \]

This form a lattice, which can be viewed as a Hasse diagram
If $\text{minsup} = 2$, the frequent itemsets are (in yellow):

- $l = \text{l}eme\text{n}$
- $p = \text{p}asta$
- $b = \text{b}read$
- $0 = \text{o}range$
- $c = \text{c}ake$

The itemsets include:

- $l$
- $p$
- $b$
- $o$
- $c$
- $lp$
- $lb$
- $lo$
- $lc$
- $pb$
- $po$
- $pc$
- $bo$
- $bc$
- $oc$
- $lpb$
- $lpo$
- $lpc$
- $lbo$
- $lbc$
- $loc$
- $pbo$
- $pbc$
- $poc$
- $boc$
- $lpbo$
- $lpbc$
- $lpoc$
- $lboc$
- $pboc$
- $lpboc$
How to find the frequent itemsets?

Two challenges:

- How to count the support of itemsets in an efficient way (not spend too much time or memory)?
- How to reduce the search space (we do not want to consider all the possibilities)?
THE APRIORI ALGORITHM
(AGRAWAL & SRIKANT, 1993/1994)

Introduction

Apriori is a famous algorithm

- which is not the most efficient algorithm,
- but has inspired many other algorithms!
- has been applied in many fields,
- has been adapted for many other similar problems.

Apriori is based on two important properties
**Apriori property:** Let there be two itemsets \( X \) and \( Y \). If \( X \subset Y \), the support of \( Y \) is less than or equal to the support of \( X \).

**Example:**
- The support of \{pasta\} is 4
- The support of \{pasta, lemon\} is 3
- The support of \{pasta, lemon, orange\} is 2

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items appearing in the transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>{pasta, lemon, bread, orange}</td>
</tr>
<tr>
<td>T2</td>
<td>{pasta, lemon}</td>
</tr>
<tr>
<td>T3</td>
<td>{pasta, orange, cake}</td>
</tr>
<tr>
<td>T4</td>
<td>{pasta, lemon, orange, cake}</td>
</tr>
</tbody>
</table>

(support is anti-monotonic)
Illustration

\[
\text{minsup} = 2
\]

frequent itemsets

Infrequent itemsets
This property is useful to reduce the search space.

Example:

\[
\text{mins}up = 2
\]
This property is useful to reduce the search space.

Example:

If « bread » is infrequent, all its supersets are infrequent.

\( \text{mins} = 2 \)
**Property 2:** Let there be an itemset $Y$. If there exists an itemset $X \subset Y$ such that $X$ is infrequent, then $Y$ is infrequent.

**Example:**
- Consider $\{\text{bread, lemon}\}$.
- If we know that $\{\text{bread}\}$ is infrequent, then we can infer that $\{\text{bread, lemon}\}$ is also infrequent.

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items appearing in the transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>{pasta, lemon, bread, orange}</td>
</tr>
<tr>
<td>T2</td>
<td>{pasta, lemon}</td>
</tr>
<tr>
<td>T3</td>
<td>{pasta, orange, cake}</td>
</tr>
<tr>
<td>T4</td>
<td>{pasta, lemon, orange, cake}</td>
</tr>
</tbody>
</table>
The Apriori algorithm

- I will now explain how the Apriori algorithm works

**Input:**
- $\text{minsup}$
- a transactional database

**Output:**
- all the frequent itemsets

Consider $\text{minsup} = 2$. 
The Apriori algorithm

**Step 1**: scan the database to calculate the support of all itemsets of size 1.

*e.g.*

- \{pasta\} \quad \text{support} = 4
- \{lemon\} \quad \text{support} = 3
- \{bread\} \quad \text{support} = 1
- \{orange\} \quad \text{support} = 3
- \{cake\} \quad \text{support} = 2
The Apriori algorithm

**Step 2:** eliminate infrequent itemsets.

e.g.

- \{pasta\} \hspace{1cm} \text{support} = 4
- \{lemon\} \hspace{1cm} \text{support} = 3
- \{bread\} \hspace{1cm} \text{support} = 1
- \{orange\} \hspace{1cm} \text{support} = 3
- \{cake\} \hspace{1cm} \text{support} = 2
The Apriori algorithm

**Step 2**: eliminate infrequent itemsets.

e.g.

- \{pasta\} \quad \text{support} = 4
- \{lemon\} \quad \text{support} = 3
- \{orange\} \quad \text{support} = 3
- \{cake\} \quad \text{support} = 2
Step 3: generate candidates of size 2 by combining pairs of frequent itemsets of size 1.

Frequent items:

- \{pasta\}
- \{lemon\}
- \{orange\}
- \{cake\}

Candidates of size 2:

- \{pasta, lemon\}
- \{pasta, orange\}
- \{pasta, cake\}
- \{lemon, orange\}
- \{lemon, cake\}
- \{orange, cake\}
The Apriori algorithm

Step 4: Eliminate candidates of size 2 that have an infrequent subset (Property 2)

(None!)

Frequent items

{pasta}
{lemon}
{orange}
{cake}

Candidates of size 2

{pasta, lemon}
{pasta, orange}
{pasta, cake}
{lemon, orange}
{lemon, cake}
{orange, cake}
The Apriori algorithm

**Step 5:** scan the database to calculate the support of remaining candidate itemsets of size 2.

Candidates of size 2

- \{pasta, lemon\} support: 3
- \{pasta, orange\} support: 3
- \{pasta, cake\} support: 2
- \{lemon, orange\} support: 2
- \{lemon, cake\} support: 1
- \{orange, cake\} support: 2
The Apriori algorithm

Step 6: eliminate infrequent candidates of size 2

Candidates of size 2

- \{pasta, lemon\} support: 3
- \{pasta, orange\} support: 3
- \{pasta, cake\} support: 2
- \{lemon, orange\} support: 2
- \{lemon, cake\} support: 1
- \{orange, cake\} support: 2
The Apriori algorithm

Step 6: eliminate infrequent candidates of size 2

Frequent itemsets of size 2:

- \{pasta, lemon\} support: 3
- \{pasta, orange\} support: 3
- \{pasta, cake\} support: 2
- \{lemon, orange\} support: 2
- \{orange, cake\} support: 2
Step 7: generate candidates of size 3 by combining frequent pairs of itemsets of size 2.

Frequent itemsets of size 2

\{pasta, lemon\}  \{pasta, orange\}  \{pasta, cake\}  \{lemon, orange\}  \{orange, cake\}

Candidates of size 3

\{pasta, lemon, orange\}  \{pasta, lemon, cake\}  \{pasta, orange, cake\}  \{lemon, orange, cake\}
The Apriori algorithm

Step 8: eliminate candidates of size 3 having a subset of size 2 that is infrequent.

Frequent itemsets of size 2

{pasta, lemon}
{pasta, orange}
{pasta, cake}
{l sesame, orange}
{orange, cake}

Candidates of size 3

{pasta, lemon, orange}
{pasta, lemon, cake}
{pasta, orange, cake}
{l sesame, orange, cake}

Because {l sesame, cake} is infrequent!
The Apriori algorithm

**Step 8**: eliminate candidates of size 3 having a subset of size 2 that is infrequent.

**Frequent itemsets of size 2**

\{pasta, lemon\}
\{pasta, orange\}
\{pasta, cake\}
\{lemon, orange\}
\{orange, cake\}

**Candidates of size 3**

\{pasta, lemon, orange\}
\{pasta, orange, cake\}

Because \{lemon, cake\} is infrequent!
Step 9: scan the database to calculate the support of the remaining candidates of size 3.

Candidates of size 2

\{\text{pasta, lemon, orange}\} \text{ support: 2}

\{\text{pasta, orange, cake}\} \text{ support: 2}
The Apriori algorithm

Step 10: eliminate infrequent candidates (none!)

Frequent itemsets of size 3

{pasta, lemon, orange} support: 2
{pasta, orange, cake} support: 2
The Apriori algorithm

**Step 11**: generate candidates of size 4 by combining pairs of frequent itemsets of size 3.

Frequent itemsets of size 3

- \{pasta, lemon, orange\}
- \{pasta, orange, cake\}

Candidates of size 4

- \{pasta, lemon, orange, cake\}
The Apriori algorithm

**Step 12:** eliminate candidates of size 4 having a subset of size 3 that is infrequent.

Frequent itemsets of size 3

\{pasta, lemon, orange\}

\{pasta, orange, cake\}

Candidates of size 4

\{pasta, lemon, orange, cake\}
Step 12: Since there is no more candidates, we cannot generate candidates of size 5 and the algorithm stops.

Candidates of size 4

\{pasta, lemon, orange, cake\}
Final result

{pasta}          support = 4
{lemon}          support = 3
{orange}         support = 3
{cake}           support = 2

{pasta, lemon}   support: 3
{pasta, orange}  support: 3
{pasta, cake}    support: 2
{lemon, orange}  support: 2
{orange, cake}   support: 2

{pasta, lemon, orange} support: 2
{pasta, orange, cake} support: 2
Technical details

Combining different itemsets can generate the same candidate.

Example:

\{A, B\} \text{ and } \{A,E\} \rightarrow \{A, B, E\}

\{B, E\} \text{ and } \{A,E\} \rightarrow \{A, B, E\}

\textbf{problem:} some candidates are generated several times!
Technical details

Combining different itemsets can generate the same candidate.

Example:

\{A, B\} and \{A, E\} \rightarrow \{A, B, E\}

\{B, E\} and \{A, E\} \rightarrow \{A, B, E\}

Solution:

• Sort items in each itemsets (e.g. by alphabetical order)
• Combine two itemsets only if all items are the same except the last one.
The Apriori property can considerably reduce the number of itemsets to be considered.

In the previous example:

- Naïve approach:
  \[2^5 - 1 = 31\] itemsets are considered

- By using the Apriori property:
  18 itemsets are considered
PERFORMANCE COMPARISON
How to evaluate this type of algorithms?

- Execution time,
- Memory used,
- **Scalability**: how the performance is influenced by the number of transactions
- Performance on different types of data:
  - real data,
  - synthetic (fake) data,
  - dense vs sparse data,…
- …
Performance (execution time)

"Chess" dataset

Execution time (s)

Note: Eclat ran out of memory at 0.8
Performance (execution time)
Performance of Apriori

The performance of Apriori depends on several factors:

- **the \textit{minsup} parameter**: the more it is set low, the larger the search space and the number of itemsets will be.
- the number of items,
- the number of transactions,
- The average transaction length.
Problems of Apriori

- can generate numerous candidates
- requires to scan the database numerous times.
- candidates may not exist in the database.
- …
A FEW OPTIMIZATIONS FOR THE APRIORI ALGORITHM

This is an advanced topic
Optimization 1

In terms of data structure:

- Store all items as integers:
  - e.g. 1 = pasta, 2 = orange, 3 = bread...

- Why?
  - it is faster to compare two integers than to compare two character strings,
  - requires less memory.
Optimization 2

To reduce the time required to calculate the support of itemsets

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items appearing in the transaction</th>
<th>sort transactions by ascending length</th>
</tr>
</thead>
<tbody>
<tr>
<td>T2</td>
<td>{pasta, lemon}</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>{pasta, orange, cake}</td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>{pasta, lemon, orange, cake}</td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td>{pasta, lemon, bread, orange}</td>
<td></td>
</tr>
</tbody>
</table>

• To calculate the support of an itemset of size \( k \), only the transactions of size \( \geq k \) are used.
Optimization 3

To reduce the time required to calculate the support of itemsets

- Replace all identical transactions by a single transactions with a weight.
Optimization 4

To reduce the time require to calculate the support of itemsets:

- Sort items in transactions according to a total order (e.g. alphabetical order).
- Utilize binary search to quickly check if an item appears in a transaction.
Optimization 5

- Store candidates in a hash tree
- To calculate the support of candidates
  - Calculate a hash value based on a transaction to determine if candidates are contained in the transaction.
Other optimizations

- Sampling and partitioning
AprioriTID: a variation

AprioriTID:
- Annotate each itemset with the ids of transactions that contain it,
- use the intersection (∩) to calculate the support of itemsets instead of reading the database.

Example →
<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items appearing in the transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>{pasta, lemon, bread, orange}</td>
</tr>
<tr>
<td>T2</td>
<td>{pasta, lemon}</td>
</tr>
<tr>
<td>T3</td>
<td>{pasta, orange, cake}</td>
</tr>
<tr>
<td>T4</td>
<td>{pasta, lemon, orange, cake}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Transactions containing the item</th>
</tr>
</thead>
<tbody>
<tr>
<td>pasta</td>
<td>T1, T2, T3, T4</td>
</tr>
<tr>
<td>lemon</td>
<td>T1, T2, T3</td>
</tr>
<tr>
<td>bread</td>
<td>T1</td>
</tr>
<tr>
<td>orange</td>
<td>T1, T3, T4</td>
</tr>
<tr>
<td>cake</td>
<td>T3, T4</td>
</tr>
<tr>
<td>item</td>
<td>transactions containing the item</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>pasta</td>
<td>T1, T2, T3, T4</td>
</tr>
<tr>
<td>lemon</td>
<td>T1, T2, T4</td>
</tr>
<tr>
<td>bread</td>
<td>T1</td>
</tr>
<tr>
<td>orange</td>
<td>T1, T3, T4</td>
</tr>
<tr>
<td>cake</td>
<td>T3, T4</td>
</tr>
</tbody>
</table>

Example: calculating the support of \{pasta, lemon\}:

\[
\text{transactions(\{pasta\}) } \cap \text{transactions(\{lemon\})} = \{T1, T2, T3, T4\} \cap \{T1, T2, T4\} = \{T1, T2, T4\}
\]

Thus \{pasta, lemon\} has a support of 3
AprioriTID_Bitset

AprioriTID_bitset:

- Same idea, except that bit vectors are used instead of lists of ids.
- This allows to calculate the intersection using the `Logical_AND`, which is often very fast.

Example ➔
<table>
<thead>
<tr>
<th>Transaction</th>
<th>Items appearing in the transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>{pasta, lemon, bread, orange}</td>
</tr>
<tr>
<td>T2</td>
<td>{pasta, lemon}</td>
</tr>
<tr>
<td>T3</td>
<td>{pasta, orange, cake}</td>
</tr>
<tr>
<td>T4</td>
<td>{pasta, lemon, orange, cake}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>item</th>
<th>transactions containing the item</th>
</tr>
</thead>
<tbody>
<tr>
<td>pasta</td>
<td>1111 (representing T1, T2, T3, T4)</td>
</tr>
<tr>
<td>lemon</td>
<td>1101</td>
</tr>
<tr>
<td>bread</td>
<td>1000</td>
</tr>
<tr>
<td>orange</td>
<td>1011</td>
</tr>
<tr>
<td>cake</td>
<td>0011</td>
</tr>
<tr>
<td>item</td>
<td>transactions containing the item</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>pasta</td>
<td>1111</td>
</tr>
<tr>
<td>lemon</td>
<td>1101</td>
</tr>
<tr>
<td>bread</td>
<td>1000</td>
</tr>
<tr>
<td>orange</td>
<td>1011</td>
</tr>
<tr>
<td>cake</td>
<td>0011</td>
</tr>
</tbody>
</table>

**Example: Calculate the support of \{pasta, lemon\} :**

\[
\text{transactions}\left(\{\text{pasta}\}\right) \cap \text{transactions}\left(\{\text{lemon}\}\right) = 1111 \ \text{LOGICAL\_AND} \ 1101 = 1101
\]

Thus \{pasta, lemon\} has a support of 3
Conclusion

This video has presented:

- The problem of frequent itemset mining
- The Apriori algorithm
- Some optimizations
References

- …