Brief Introduction to Algorithmic Data Analysis in English

Esther Galbrun
Autumn 2020
Part I

Frequent Itemset Mining
Problem
Discover items that often co-occur in a dataset

Classical setting: *Shopping basket data*

- Each product of the supermarket is an item
- Record customer transactions as sets of items
- Identify products that are often bought together
  
  **Frequent itemset** \{butter, bread, ham, pickles\}

- Extract rules that capture typical buying behaviour
  
  **Association rules** \{bread, ham\} $\Rightarrow$ \{butter, pickles\}

- Insights for marketing and shelf placement
Frequent Itemset Mining

Discover items that often co-occur in a dataset

**Shopping basket data**  Customer transactions
Identify products often bought together

**Text mining**  Bag of word model
Identify co-occurring terms and keywords

**More complex data types**  (spatio-)temporal data, graph data

**Other analysis tasks**  Building block for clustering, classification, outlier detection
A pizzeria offers to compose your pizza by freely choosing ingredients among *ham, jalapeno, mozzarella, olives and tuna*. To put together a menu, the pizzaiolo would like to know what are favorite combinations.
The database $\mathcal{T}$ is a collection of sets, called transactions, from a universe $U$ of items:

$$\mathcal{T} = \{T_1, T_2, \ldots, T_n\}, \text{ where } T_k \subseteq U, \forall k \in [1, n]$$

The total number of items is $m = |U|$.

If we fix an order over $U$, each transaction can be represented as a binary vector of size $m$.

Then, the database can be represented as a binary matrix with $n$ rows and $m$ columns.

Each transaction has a unique identifier, its $tid$. 
Pizzeria example

The universe of items is the set of five ingredients
{ham, jalapeno, mozzarella, olives, tuna}

For short, \( U = \{h, j, m, o, t\} \)

Each pizza constitutes a transaction, represented by the corresponding set of ingredients
For instance, a ham and mozzarella pizza is represented as \( T = \{h, m\} \), also simply denoted \( hm \)

Ordering the items alphabetically according to corresponding ingredient names, this pizza is represented by the binary vector \( \langle 1, 0, 1, 0, 0 \rangle \), also simply written 10100
### Pizzeria example

The database then records all pizzas sold

<table>
<thead>
<tr>
<th>tids</th>
<th>pizzas</th>
<th>sets</th>
<th>matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>ham mozzarella olives</td>
<td>{h, m, o}</td>
<td>■■■■■</td>
</tr>
<tr>
<td>2)</td>
<td>mozzarella</td>
<td>{m}</td>
<td>■■■■■</td>
</tr>
<tr>
<td>3)</td>
<td>jalapeno mozzarella</td>
<td>{j, m}</td>
<td>■■■■■</td>
</tr>
<tr>
<td>4)</td>
<td>ham jalapeno mozzarella olives</td>
<td>{h, j, m, o}</td>
<td>■■■■■</td>
</tr>
<tr>
<td>5)</td>
<td>ham jalapeno mozzarella olives</td>
<td>{h, j, m, o}</td>
<td>■■■■■</td>
</tr>
<tr>
<td>6)</td>
<td>ham</td>
<td>{h}</td>
<td>■■■■■</td>
</tr>
<tr>
<td>7)</td>
<td>ham jalapeno mozzarella tuna</td>
<td>{h, j, m, t}</td>
<td>■■■■■</td>
</tr>
<tr>
<td>8)</td>
<td>mozzarella</td>
<td>{m}</td>
<td>■■■■■</td>
</tr>
<tr>
<td>9)</td>
<td>olives</td>
<td>{o}</td>
<td>■■■■■</td>
</tr>
<tr>
<td>10)</td>
<td>ham jalapeno mozzarella olives</td>
<td>{h, j, m, o, t}</td>
<td>■■■■■</td>
</tr>
<tr>
<td>11)</td>
<td>ham mozzarella tuna</td>
<td>{h, m, t}</td>
<td>■■■■■</td>
</tr>
<tr>
<td>12)</td>
<td>ham mozzarella</td>
<td>{h, m}</td>
<td>■■■■■</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
An **itemset** $I$ is a set of items, i.e. $I \subseteq U$

A **$k$-itemset** is an itemset that contains exactly $k$ items, i.e. such that $|I| = k$

The **support set** of an itemset $I$ in $\mathcal{T}$ is the set of transactions from $\mathcal{T}$ that contain $I$

$$\text{supp}_\mathcal{T}(I) = \{T \in \mathcal{T}, I \subseteq T\}$$

We call $|\text{supp}_\mathcal{T}(I)|$ the **absolute support** of $I$ in $\mathcal{T}$ and $|\text{supp}_\mathcal{T}(I)| / |\mathcal{T}|$ its **fractional support**

We denote $\text{supp} \%_\mathcal{T}(I)$ the fractional support given as a percentage, i.e.

$$\text{supp} \%_\mathcal{T}(I) = 100 \cdot \frac{|\text{supp}_\mathcal{T}(I)|}{|\mathcal{T}|}$$
An itemset $I$ is a set of items, i.e. $I \subseteq U$

A $k$-itemset is an itemset that contains exactly $k$ items, i.e. such that $|I| = k$

The support set of an itemset $I$ in $\mathcal{T}$ is the set of transactions from $\mathcal{T}$ that contain $I$

$$\text{supp}_\mathcal{T}(I) = \{T \in \mathcal{T}, I \subseteq T\}$$

We call $|\text{supp}_\mathcal{T}(I)|$ the absolute support of $I$ in $\mathcal{T}$ and $|\text{supp}_\mathcal{T}(I)| / |\mathcal{T}|$ its fractional support

! There are variations in the use of support terminology
! The database is often left out from the notation, as it is clear from the context
In this database, for itemset $l = \{t\}$

$$supp(l) = \{7, 10, 11\}$$

$$|supp(l)| = 3$$

$$supp\% (l) = 25$$

<table>
<thead>
<tr>
<th>tid</th>
<th>set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>{h, m, o}</td>
</tr>
<tr>
<td>2)</td>
<td>{m}</td>
</tr>
<tr>
<td>3)</td>
<td>{j, m}</td>
</tr>
<tr>
<td>4)</td>
<td>{h, j, m, o}</td>
</tr>
<tr>
<td>5)</td>
<td>{h, j, m, o}</td>
</tr>
<tr>
<td>6)</td>
<td>{h}</td>
</tr>
<tr>
<td>7)</td>
<td>{h, j, m, t}</td>
</tr>
<tr>
<td>8)</td>
<td>{m}</td>
</tr>
<tr>
<td>9)</td>
<td>{o}</td>
</tr>
<tr>
<td>10)</td>
<td>{h, j, m, o, t}</td>
</tr>
<tr>
<td>11)</td>
<td>{h, m, t}</td>
</tr>
<tr>
<td>12)</td>
<td>{h, m}</td>
</tr>
</tbody>
</table>
### Pizzeria example

<table>
<thead>
<tr>
<th>tid</th>
<th>set</th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>{h, m, o}</td>
</tr>
<tr>
<td>2)</td>
<td>{m}</td>
</tr>
<tr>
<td>3)</td>
<td>{j, m}</td>
</tr>
<tr>
<td>4)</td>
<td>{h, j, m, o}</td>
</tr>
<tr>
<td>5)</td>
<td>{h, j, m, o}</td>
</tr>
<tr>
<td>6)</td>
<td>{h}</td>
</tr>
<tr>
<td>7)</td>
<td>{h, j, m, t}</td>
</tr>
<tr>
<td>8)</td>
<td>{m}</td>
</tr>
<tr>
<td>9)</td>
<td>{o}</td>
</tr>
<tr>
<td>10)</td>
<td>{h, j, m, o, t}</td>
</tr>
<tr>
<td>11)</td>
<td>{h, m, t}</td>
</tr>
<tr>
<td>12)</td>
<td>{h, m}</td>
</tr>
</tbody>
</table>

In this database, for itemset $I = \{h, m\}$

$supp(I) = \{1, 4, 5, 7, 10, 11, 12\}$

$|supp(I)| = 7$

$supp\% (I) = 58.33$
Frequent Itemset Mining

Given a set of transactions $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$, where each transaction $T_i$ is a subset of items from $U$, and a minimum support threshold $\sigma$, determine all itemsets $I$ that occur as a subset of at least $\sigma$ transactions in $\mathcal{T}$. 
## Pizzeria example

### Enumerate all distinct pizzas

<table>
<thead>
<tr>
<th></th>
<th>count</th>
<th>tids</th>
</tr>
</thead>
<tbody>
<tr>
<td>hmo</td>
<td>65</td>
<td>{1...}</td>
</tr>
<tr>
<td>m</td>
<td>74</td>
<td>{2, 8...}</td>
</tr>
<tr>
<td>jm</td>
<td>28</td>
<td>{3...}</td>
</tr>
<tr>
<td>hjmo</td>
<td>47</td>
<td>{4, 5...}</td>
</tr>
<tr>
<td>h</td>
<td>74</td>
<td>{6...}</td>
</tr>
<tr>
<td>hjmt</td>
<td>30</td>
<td>{7...}</td>
</tr>
<tr>
<td>o</td>
<td>178</td>
<td>{9...}</td>
</tr>
<tr>
<td>hjmot</td>
<td>93</td>
<td>{10...}</td>
</tr>
<tr>
<td>hmt</td>
<td>49</td>
<td>{11...}</td>
</tr>
<tr>
<td>hm</td>
<td>96</td>
<td>{12...}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>count</th>
<th>tids</th>
</tr>
</thead>
<tbody>
<tr>
<td>hj</td>
<td>16</td>
<td>{...}</td>
</tr>
<tr>
<td>hjm</td>
<td>42</td>
<td>{...}</td>
</tr>
<tr>
<td>hjo</td>
<td>8</td>
<td>{...}</td>
</tr>
<tr>
<td>hjot</td>
<td>0</td>
<td>{}</td>
</tr>
<tr>
<td>hjt</td>
<td>4</td>
<td>{...}</td>
</tr>
<tr>
<td>hmot</td>
<td>108</td>
<td>{...}</td>
</tr>
<tr>
<td>ho</td>
<td>43</td>
<td>{...}</td>
</tr>
<tr>
<td>hot</td>
<td>28</td>
<td>{...}</td>
</tr>
<tr>
<td>ht</td>
<td>7</td>
<td>{...}</td>
</tr>
<tr>
<td>j</td>
<td>29</td>
<td>{...}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>count</th>
<th>tids</th>
</tr>
</thead>
<tbody>
<tr>
<td>jmo</td>
<td>7</td>
<td>{...}</td>
</tr>
<tr>
<td>jmot</td>
<td>9</td>
<td>{...}</td>
</tr>
<tr>
<td>jmt</td>
<td>3</td>
<td>{...}</td>
</tr>
<tr>
<td>jo</td>
<td>16</td>
<td>{...}</td>
</tr>
<tr>
<td>jot</td>
<td>3</td>
<td>{...}</td>
</tr>
<tr>
<td>jt</td>
<td>14</td>
<td>{...}</td>
</tr>
<tr>
<td>mo</td>
<td>20</td>
<td>{...}</td>
</tr>
<tr>
<td>mot</td>
<td>27</td>
<td>{...}</td>
</tr>
<tr>
<td>mt</td>
<td>13</td>
<td>{...}</td>
</tr>
<tr>
<td>ot</td>
<td>17</td>
<td>{...}</td>
</tr>
<tr>
<td>t</td>
<td>8</td>
<td>{...}</td>
</tr>
</tbody>
</table>

---

**UEF//School of Computing**

**BADA: FIM**
Pizzeria example: Enumerating all distinct pizzas

\[
\begin{array}{c}
\emptyset \\
0 \\
\{
\end{array}
\]

\[
\begin{array}{cccccc}
\text{h} & \text{j} & \text{m} & \text{o} & \text{t} \\
74 & 29 & 74 & 178 & 8 \\
{6\ldots} & {\ldots} & {2,8\ldots} & {9\ldots} & {\ldots}
\end{array}
\]

\[
\begin{array}{cccccccc}
\text{hj} & \text{hm} & \text{ho} & \text{ht} & \text{jm} & \text{jo} & \text{jt} & \text{mo} & \text{mt} & \text{ot} \\
16 & 96 & 43 & 7 & 28 & 16 & 14 & 20 & 13 & 17 \\
{\ldots} & {12\ldots} & {\ldots} & {3\ldots} & {\ldots} & {\ldots} & {\ldots} & {\ldots} & {\ldots} & {\ldots}
\end{array}
\]

\[
\begin{array}{cccccccc}
\text{hjm} & \text{hjo} & \text{hjt} & \text{hmo} & \text{hmt} & \text{hot} & \text{jmo} & \text{jmt} & \text{jot} & \text{mot} \\
42 & 8 & 4 & 65 & 49 & 28 & 7 & 3 & 3 & 27 \\
{\ldots} & {\ldots} & {\ldots} & {1\ldots} & {11\ldots} & {\ldots} & {\ldots} & {\ldots} & {\ldots} & {\ldots}
\end{array}
\]

\[
\begin{array}{cccccccc}
\text{hjmo} & \text{hjmt} & \text{hjot} & \text{hmot} & \text{jmot} \\
47 & 30 & 0 & 108 & 9 \\
{4,5\ldots} & {7\ldots} & {\ldots} & {\ldots} & {\ldots}
\end{array}
\]

\[
\begin{array}{c}
\text{hjmot} \\
93 \\
{10\ldots}
\end{array}
\]
Pizzeria example: Aggregating supports

\[
\begin{array}{cccccc}
\emptyset & 0 \\
\{0\} & \\
\h & 74 \\
\{6\ldots\} & \{\ldots\} & \{2,8\ldots\} & \{9\ldots\} & \{\ldots\} \\
\j & 29 \\
\{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\m & 74 \\
\{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\o & 178 \\
\{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\t & 8 \\
\{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\h \j & 16 \\
\{\ldots\} & \{12\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\h \m & 96 \\
\{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\h \o & 43 \\
\{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\h \t & 7 \\
\{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\j \m & 28 \\
\{3\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\j \o & 16 \\
\{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\j \t & 14 \\
\{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\m \o & 20 \\
\{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\m \t & 13 \\
\{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\o \t & 17 \\
\{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\h \j \m & 42 \\
\{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\h \j \o & 8 \\
\{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\h \j \t & 4 \\
\{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\h \j \m \o & 65 \\
\{1\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\h \j \m \t & 49 \\
\{11\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\h \j \o \t & 28 \\
\{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\h \j \m \t \o & 7 \\
\{1\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\h \j \m \o \t & 3 \\
\{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\h \j \m \o \t \o & 108 \\
\{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\h \j \m \o \t \m & 9 \\
\{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\h \j \m \o \t \o \t & 93 \\
\{10\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} & \{\ldots\} \\
\end{array}
\]
Pizzeria example: Aggregating supports

The diagram illustrates a hierarchical structure representing supports in a pizzeria example. The nodes represent sets, and the numbers indicate the count of orders for each set.

For instance:
- hjm: 42
- hjot: 0
- hjmot: 93

The sets are connected in a way that each set is a subset of the one above it, forming a tree-like structure with the root node at the top and the leaf nodes at the bottom.

The numbers next to the nodes represent the support count, which is a measure of how often a set of items is ordered together.
Support properties

Monotonicity of support
The support of every subset \( J \) of \( I \) is at least equal to that of the support of itemset \( I \)

\[
\forall J \subseteq I, \quad \text{supp}(I) \subseteq \text{supp}(J)
\]

and hence

\[
|\text{supp}(I)| \leq |\text{supp}(J)|
\]

Downward closure property
Every subset of a frequent itemset is also frequent
Pizzeria example: Lattice of ingredient combinations
Pizzeria example: Frequent ingredient combinations

for minimum support
\[ \sigma = 116 = 0.10 \cdot |\mathcal{T}| \]
Pizzeria example: Frequent ingredient combinations

for minimum support

\[
\sigma = 289 = 0.25 \cdot |\mathcal{T}| 
\]
Maximal and closed itemsets

A frequent itemset $I$ is **maximal** at a given minimum support level $\sigma$, if it is frequent, and no superset of it is frequent.

An itemset $I$ is **closed**, if none of its supersets have exactly the same support count as $I$.

**Condensed representations**
Knowledge of maximal frequent itemsets allows to reconstruct the set of frequent itemsets, but not their supports.
Knowledge of closed frequent itemsets allows to also recompute the supports.
Pizzeria example: Frequent and maximal itemsets

For minimum support $\sigma = 116 = 0.10 \cdot |T|$
Pizzeria example: Frequent and maximal itemsets

\[ \sigma = 289 = 0.25 \cdot |T| \]
Algorithms
Algorithms for mining frequent itemsets

Support counting is expensive

Explore the space of itemsets by increasing lengths, i.e. *level-wise enumeration*

Avoid generating itemsets twice by using a canonical order

Exploit the *downward closure property* to prune itemsets
Level-wise enumeration: *Apriori* algorithm

\[ k \leftarrow 1 \]
\[ \mathcal{F}_k \leftarrow \{ \text{all frequent singleton itemsets} \} \]

**while** \( \mathcal{F}_k \neq \emptyset \) **do**

- Generate \( \mathcal{C}_{k+1} \) by extending itemsets from \( \mathcal{F}_k \)
- Prune itemsets that violate downward closure

\[ \mathcal{F}_{k+1} \leftarrow \{ S \in \mathcal{C}_{k+1}, \text{supp}_D(S) \geq \theta \} \]
\[ k \leftarrow k + 1 \]

**return** \( \bigcup \mathcal{F}_i \)
Candidate generation

\[ k \leftarrow 1 \]
\[ \mathcal{F}_k \leftarrow \{ \text{all frequent singleton itemsets} \} \]
\[ \text{while } \mathcal{F}_k \neq \emptyset \text{ do} \]
\[ \quad \text{Generate } \mathcal{C}_{k+1} \text{ by extending itemsets from } \mathcal{F}_k \]
\[ \quad \text{Prune itemsets that violate downward closure} \]
\[ \quad \mathcal{F}_{k+1} \leftarrow \{ \mathcal{S} \in \mathcal{C}_{k+1}, \text{supp}_D(\mathcal{S}) \geq \theta \} \]
\[ \quad k \leftarrow k + 1 \]
\[ \text{return } \bigcup_i \mathcal{F}_i \]
Level-wise enumeration: *Apriori* algorithm

Candidate pruning

\[
k \leftarrow 1 \\
\mathcal{F}_k \leftarrow \{\text{all frequent singleton itemsets}\} \\
\textbf{while } \mathcal{F}_k \neq \emptyset \textbf{ do} \\
\quad \text{Generate } \mathcal{C}_{k+1} \text{ by extending itemsets from } \mathcal{F}_k \\
\quad \text{Prune itemsets that violate downward closure} \\
\quad \mathcal{F}_{k+1} \leftarrow \{\mathcal{S} \in \mathcal{C}_{k+1}, \text{supp}_D(\mathcal{S}) \geq \theta\} \\
\quad k \leftarrow k + 1 \\
\textbf{return } \bigcup_i \mathcal{F}_i
\]
Level-wise enumeration: *Apriori* algorithm

**Support counting**

\[
k \leftarrow 1 \\
\mathcal{F}_k \leftarrow \{\text{all frequent singleton itemsets}\} \\
\textbf{while } \mathcal{F}_k \neq \emptyset \textbf{ do} \\
\quad \text{Generate } \mathcal{C}_{k+1} \text{ by extending itemsets from } \mathcal{F}_k \\
\quad \text{Prune itemsets that violate downward closure} \\
\quad \mathcal{F}_{k+1} \leftarrow \{S \in \mathcal{C}_{k+1}, \text{supp}_D(S) \geq \theta\} \\
\quad k \leftarrow k + 1 \\
\textbf{return } \bigcup_i \mathcal{F}_i \]
Pizzeria example: Apriori algorithm

Enumerate singleton itemsets

\[
\emptyset
\]

\[
\begin{align*}
h & \quad j & \quad m & \quad o & \quad t
\end{align*}
\]

for \( \sigma = 289 \)

1156
Pizzeria example: *Apriori* algorithm

Count supports

![Count supports diagram](image)

for $\sigma = 289$
Frequent singleton itemsets

for $\sigma = 289$
Pizzeria example: Apriori algorithm

Generate candidates itemsets of length 2

for $\sigma = 289$
Pizzeria example: *Apriori* algorithm

Count supports

![Support Count Diagram]

for $\sigma = 289$
Pizzeria example: *Apriori* algorithm

Frequent itemsets of length up to 2

for $\sigma = 289$
Pizzeria example: *Apriori* algorithm

Generate candidates itemsets of length 3

for $\sigma = 289$
Pizzeria example: Apriori algorithm

Prune candidates

for $\sigma = 289$
Pizzeria example: *Apriori* algorithm

Count supports

![Apriori algorithm diagram]

for $\sigma = 289$
Pizzeria example: *Apriori* algorithm

Frequent itemsets of length up to 3

```
for σ = 289
```
Generate candidates itemsets of length 4

for $\sigma = 289$
Prune candidates

for $\sigma = 289$
Pizzeria example: *Apriori* algorithm

Frequent itemsets of length up to 4

for \( \sigma = 289 \)
Pizzeria example: Enumeration tree

Items ordered alphabetically, prefix growth

for $\sigma = 289$
Pizzeria example: Enumeration tree

Items ordered by decreasing frequency, prefix growth

for $\sigma = 289$
Items ordered by increasing frequency, prefix growth

for $\sigma = 289$
Support counting is expensive

According to the monotonicity of support

\[ \forall J \subseteq I, \text{ supp}(I) \subseteq \text{ supp}(J) \]

Make support counting more efficient

• Prune irrelevant transactions
• Reuse support counting from previous steps

Recursively project the database down the enumeration tree
Vertical apriori algorithm

\[ k \leftarrow 1 \]
\[ \mathcal{F}_k \leftarrow \{ \text{all frequent singleton itemsets} \} \]
Generate \( \text{tid} \) list for each frequent singleton itemsets

\textbf{while} \( \mathcal{F}_k \neq \emptyset \) \textbf{do}

Generate \( \mathcal{C}_{k+1} \) by joining pairs of itemsets from \( \mathcal{F}_k \)
Prune itemsets that violate downward closure
Generate \( \text{tid} \) list for each candidate by intersecting tid lists of associated pair of \( k \)-itemsets

\[ \mathcal{F}_{k+1} \leftarrow \{ \mathcal{S} \in \mathcal{C}_{k+1}, \supp_D(\mathcal{S}) \geq \theta \} \]
\[ k \leftarrow k + 1 \]

\textbf{return} \( \bigcup_i \mathcal{F}_i \)
**Vertical apriori algorithm**

Vertical database representation

\[ k \leftarrow 1 \]
\[ \mathcal{F}_k \leftarrow \{ \text{all frequent singleton itemsets} \} \]

Generate *tid* list for each frequent singleton itemsets

**while** \( \mathcal{F}_k \neq \emptyset \) **do**

Generate \( \mathcal{C}_{k+1} \) by joining pairs of itemsets from \( \mathcal{F}_k \)

Prune itemsets that violate downward closure

Generate *tid* list for each candidate by intersecting tid lists of associated pair of \( k \)-itemsets

\[ \mathcal{F}_{k+1} \leftarrow \{ \mathcal{S} \in \mathcal{C}_{k+1}, \text{supp}_D(\mathcal{S}) \geq \theta \} \]

\[ k \leftarrow k + 1 \]

**return** \( \bigcup_i \mathcal{F}_i \)
Tid lists

- Allow to compute supports faster
- Require memory space for storage

Use dedicated data structures that support efficient counting
The **FP-tree** is a compact representation of the database

- Extract conditional projected database for a given suffix
- Update counts efficiently

**FP-growth** is a recursive suffix-based pattern growth algorithm
Construction of the FP-tree

Inserting transactions (items sorted by decreasing frequency)

Step# 1
Transaction mho
Construction of the FP-tree

Inserting transactions (items sorted by decreasing frequency)

Step# 2
Transaction m
Construction of the FP-tree

Inserting transactions (items sorted by decreasing frequency)

Step# 3
Transaction mj
Construction of the FP-tree

Inserting transactions (items sorted by decreasing frequency)

Step#  4
Transaction  mhoj
Construction of the FP-tree

Inserting transactions (items sorted by decreasing frequency)

Step# 5
Transaction mhoj
Construction of the FP-tree

Inserting transactions (items sorted by decreasing frequency)

Step# 6
Transaction h
Construction of the FP-tree

Inserting transactions (items sorted by decreasing frequency)

Step# 7
Transaction mhtj
Construction of the FP-tree

Inserting transactions (items sorted by decreasing frequency)

Step# 8
Transaction m
Construction of the FP-tree

Inserting transactions (items sorted by decreasing frequency)

Step# 9
Transaction 0
Construction of the FP-tree

Inserting transactions (items sorted by decreasing frequency)

Step# 10
Transaction mhotj
Construction of the FP-tree

Inserting transactions (items sorted by decreasing frequency)

Step# 11
Transaction mht
Construction of the FP-tree

Inserting transactions (items sorted by decreasing frequency)

```
Step# 12
Transaction mh
```
Construction of the FP-tree

Inserting transactions (items sorted by decreasing frequency)

Step# 100
Transaction mho
Construction of the FP-tree

Inserting transactions (items sorted by decreasing frequency)

Step# 101
Transaction mot
Construction of the FP-tree

Inserting transactions (items sorted by decreasing frequency)

Step# 1155
Transaction ho
Construction of the FP-tree

Inserting transactions (items sorted by decreasing frequency)
Suffix-based pattern growth with the FP-tree

Recursive pattern growth

for $\sigma = 289$
Suffix-based pattern growth with the FP-tree

Recursive pattern growth

for $\sigma = 289$

Suffix $m$
Suffix-based pattern growth with the FP-tree

Recursive pattern growth

for $\sigma = 289$
Suffix-based pattern growth with the FP-tree

Recursive pattern growth

for $\sigma = 289$

Suffix $h$
Suffix-based pattern growth with the FP-tree

Recursive pattern growth

for $\sigma = 289$

Suffix mh
Suffix-based pattern growth with the FP-tree

Recursive pattern growth

for $\sigma = 289$
Suffix-based pattern growth with the FP-tree

Recursive pattern growth

for \( \sigma = 289 \)

Suffix  o
Suffix-based pattern growth with the FP-tree

Recursive pattern growth

for $\sigma = 289$
Suffix mo
Suffix-based pattern growth with the FP-tree

Recursive pattern growth

![Diagram of recursive pattern growth]

for $\sigma = 289$

Suffix ho
Suffix-based pattern growth with the FP-tree

Recursive pattern growth

for $\sigma = 289$
Suffix mho
Suffix-based pattern growth with the FP-tree

Recursive pattern growth

for $\sigma = 289$

Suffix  t
Recursive pattern growth

for $\sigma = 289$

Suffix mt
Suffix-based pattern growth with the FP-tree

Recursive pattern growth

for $\sigma = 289$
Suffix $ht$
Suffix-based pattern growth with the FP-tree

Recursive pattern growth

for $\sigma = 289$

Suffix $j$

UEF//School of Computing    BADA:FIM
Pizzeria example: Enumeration tree

Items ordered by decreasing frequency, suffix growth

```
∅ 1156
 ø
  m 711
   ∅ 710
    h 669
     ∅ 413
      t 349
       ∅ 319
        ho 392
          ∅ 332
           mt 376
              ∅ 313
               mh 530
                  ∅ 313
                   mho
```
Association rules
Association rules

Frequent itemsets can be used to generate association rules

Classical setting: *Shopping basket data*

- Identify products that are often bought together
  
  Frequent itemset \{butter, bread, ham, pickles\}

- Extract rules that capture typical buying behaviour
  
  Association rules \{bread, ham\} \Rightarrow \{butter, pickles\}

- Insights for marketing and shelf placement
Frequent itemsets can be used to generate association rules.

Consider two itemsets $X$ and $Y$ such that

$$X \subset U, \quad \emptyset \neq Y \subseteq U, \quad X \cap Y = \emptyset$$

The confidence of the association rule $X \Rightarrow Y$ is the \textit{conditional probability} that a transaction contains $X \cup Y$ given that it contains $X$

$$\text{conf}(X \Rightarrow Y) = \frac{|\text{supp}(X \cup Y)|}{|\text{supp}(X)|}$$

$X$ and $Y$ are called the \textit{antecedent} and \textit{consequent} of the rule, respectively.
Association rules

$X \Rightarrow Y$ is an association rule at minimum support $\sigma$ and minimum confidence $\gamma$ if

$$\text{supp}(X \cup Y) \geq \sigma \quad \text{and} \quad \text{conf}(X \Rightarrow Y) \geq \gamma$$
1. Mine all the frequent itemsets for minimum support $\sigma$

2. Split the frequent itemsets into association rules of minimum confidence $\gamma$

**Monotonicity of confidence**

Let $X_a$, $X_b$ and $l$ be itemsets such that $X_a \subseteq X_b \subseteq l$, then

$$\text{conf}(X_b \Rightarrow l \setminus X_b) \geq \text{conf}(X_a \Rightarrow l \setminus X_a)$$