LINFO2364 Mining Patterns in Data

Issambre L'Hermite Dumont

This summary may not be up-to-date, the newer version is available at this address:
https://github.com/SimonDesmidt/Syntheses

Academic year 2025-2026 - Q2

UCLouvain

Contents

Introduction

In our data-driven world, the ability to extract meaningful information from vast datasets is crucial. Understanding all the aspect of this discipline is essential to derive those meaningful information, and this is the goal of this course. First, we need to define some key concepts.

1.1 Definitions

Definition 1.1. A pattern is a recurring structure in a dataset.

Patterns can be simple or complex, relevant or irrelevant. Their advantages is that they are interpretable. When found, relevant patterns can be used to make predictions, to understand the underlying structure of the data, and to make informed decisions.

Definition 1.2. Data mining is the process of discovering interesting patterns, models, and other kinds of knowledge in large data sets.

1.1.1 Type of data

We can mine data out of various types of structure of data:

- **Tabular data**: Data is organized in rows and columns. Example: spreadsheets, databases.
- **Sequences**: Data points are ordered in a sequence. Example: DNA sequences, text data.
- **Graphs, trees, networks**: Data is represented as nodes and edges. Example: social networks, web graphs.

Those structures can be discretes, continuous, enumerable data, etc. Those structures, can be combined to form more complex data types. And they can be highly structured, semi-structured, or unstructured.

Definition 1.3. Highly structured data are relational databases, with uniform record or table-like structures, with a fixed set of well-defined attributes. This is rarely the case in real-world data.

Definition 1.4. Semi-structured data are not as structured as in relational databases, but presents some structure with clearly defined semantic meaning. For example:

- Transactional dataset: structured into transactions, but each transaction is an unstructured set of values
- Sequence data set: unstructured collection of ordered sequences of values
- Graphs: set of nodes connected by a set of edges, with edges labelled given some semantic

Definition 1.5. Unstructured data have no predefined structure or organization. For example: text documents, images, audio files, videos.

Those requires advanced techniques to extract patterns, like deep learning or domain-specific methods. We can also categorize data based on the way they are generated:

- **Stored data**: Data is collected and stored in a finite set.
- **Streamed data**: Data is continuously generated and updated over time. Example: video surveillance, etc.

1.1.2 Types of data mining

Techniques and algorithms may vary depending on the data but also on way we will use mined patterns. We can categorize data mining tasks into two main types:

- **Descriptive data mining**: finds patterns that characterize the properties of the data.
- **Predictive data mining**: finds patterns that can be used to make predictions by induction.

We can identify patterns by multiples ways:

- **Frequent patterns**: patterns that identify a recurring structure in the data.
- **Associations patterns**: patterns that show rules of implications between different attributes
- **Correlations**: patterns that has positive or negative correlation between different attributes.

We can uses patterns for predictions:

- **Classification**: assign new data to classes based on similarities with historic data.
- **Regression**: predict numerical values based on the new data and the historic data.
- **Feature selection**: identify the most relevant features.

Preprocessing

To analyse data, and retrieve meaningful patterns, data often needs to be preprocessed. This step is crucial as raw data is often incomplete, inconsistent, and noisy. Preprocessing improves the quality of the data and the efficiency of the mining algorithms.

2.1 Useful statistical values

Definition 2.1. The mean (or average) of a dataset is the sum of all values divided by the number of values.

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{2.1}$$

Definition 2.2. The midrange is the average of the largest and smallest values in the set.

Definition 2.3. The variance of a dataset measures is defined like this:

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 \tag{2.2}$$

Definition 2.4. The standard deviation of a dataset is the square root of the variance:

$$\sigma = \sqrt{\sigma^2} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{2.3}$$

Definition 2.5. The mode for a set of data is the value that occurs most frequently. When there are multiple values with the same highest frequency, the dataset can be unimodal, bimodal, trimodal or multimodal.

Definition 2.6. Quantiles are points taken at regular intervals of a data distribution, dividing it into essentially equal-sized consecutive sets. The kth q-quantile for a given data distribution is the value x such that at most $\frac{k}{q}$ of the data values are less than x and at most $\frac{q-k}{q}$ of the data values are more than x, where k is an integer such that $0 < k < q$. There are $q - 1$ q-quantiles.

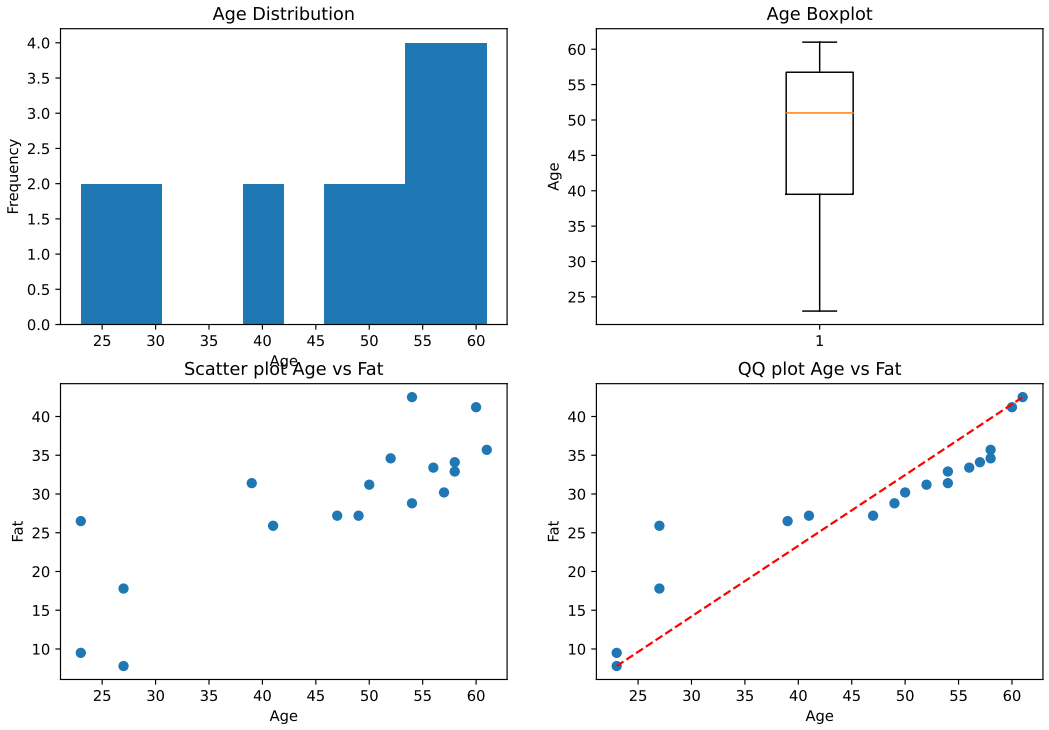
Usually we use quartiles ($q = 4$), where Q_2 is the median, or percentiles ($q = 100$).

2.2 Data plot

Plotting the data can be useful to observe some characteristics of the dataset. we can represent the data under multiples forms:

- **Histogram** is a bar chart representing the count of values present in each bucket. The buckets can be defined for each values, for ranges of values (equal-width) or for buckets containing the same number of values (equal-frequency).
- **Boxplots** are a popular way of visualizing a distribution. Typically, the ends of the box are at the quartiles so that the box length is the interquartile range. The median is marked by a line within the box. Two lines (called whiskers) outside the box extend to the smallest (minimum) and largest (maximum) observations. To highlight outliers, the whiskers are usually extended to extreme low and high observations only if these values are less than 1.5 times the interquartile range beyond the quartiles. Otherwise, the whiskers terminate at the most extreme observations occurring within 1.5 times the interquartile range.
- **Scatter plot**, each pair of values is treated as a pair of coordinates in an algebraic sense and plotted as points in the plane. It helps visually determine whether there is a relationship, pattern, or trend between the two numeric attributes.
- **Quantile-quantile plot**, or q-q plots are graphs that plot the quantiles of one univariate distribution against the corresponding quantiles of another.

Here is an example of the four plots for the same dataset:



2.3 Distances and similarities

To obtain relation between data, we can use distance and similarity measures. Distance measures the dissimilarity between two data points, while similarity measures the closeness between them. The choice of distance or similarity measure depends on the type of data and the specific problem at hand.

Definition 2.7. The norm of a vector x is:

$$||x|| = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{2.4}$$

Definition 2.8. The Minkowski distance between two vectors x and y is:

$$d(x,y) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p} \tag{2.5}$$

It is a generalization of the Euclidean distance ($p = 2$) and the Manhattan distance ($p = 1$).

Definition 2.9. The supremum distance (Minkowski when $p \to \infty$) between two vectors x and y is:

$$d(x,y) = \max_i |x_i - y_i| \tag{2.6}$$

So two vectors are similar if their distances is close to 0.

Definition 2.10. The cosine similarity between two vectors x and y is:

$$s(x,y) = \frac{x \cdot y}{||x|| \cdot ||y||} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \cdot \sqrt{\sum_{i=1}^{n} y_i^2}} \tag{2.7}$$

Here two vectors are similar if their cosine similarity is close to 1, and dissimilar if it is close to 0.

2.4 Preprocessing techniques

First we can smooth data using binning methods. Binning methods smooth a sorted data value by consulting its neighborhood. We can smooth using bins by partitioning the sorted data into a number of equal-frequency or equal-width bins, and then replacing each data value in a bin by the mean, median, or boundary values of the bin. Another type of method to manipulate data is to use normalization. There exists multiple normalization techniques:

- **Vector normalization**: scales a vector x by it's norm to have a unit norm vector:

$$x' = \frac{x}{||x||} \tag{2.8}$$

- **Min-max normalization:** scales the vector linearly to fit in a specific range $[new\_min, new\_max]$

$$x' = \frac{(x - min(x))(new\_max - new\_min)}{max(x) - min(x)} + new\_min \tag{2.9}$$

- **Z-score normalization:** (or standardization) rescales the data to have a mean of 0 and a standard deviation of 1:

$$x' = \frac{x - \bar{x}}{\sigma} \tag{2.10}$$

- **Normalization by decimal scaling**:

$$x' = \frac{x}{10^j} \quad \text{where } j \text{ is the smallest integer such that } \max(|x'|) < 1 \tag{2.11}$$

  j can be computed as $j = \lceil \log_{10}(max(|x|)) + \epsilon \rceil$ with ϵ a small value to assure $\max(|x'|) < 1$.

For missing values, we can (last 3 methods introduce bias in the data):

- Ignore the tuple: usually done when the class label is missing or when multiple values are missing in the same tuple.
- Fill the missing value manually: time-consuming if lots of missing data
- Use a global constant (Unknown or $-\infty$): mining algorithms might mistakenly think that they form an interesting concept by having this "missing" value in common
- Use a measure of the central tendency of the attribute (e.g., mean or median)
- Use the measure of the central tendency for all samples belonging to the same class
- Use the most probable value to fill: using regression, inference-based tools, or decision tree induction.

We can also reduce the size of the data using sampling:

- **SRSWOR** (Simple random sample without replacement of size s): this sampling is created by drawing samples from D, and every time a sample is drawn, it is not to be placed back into the data set D.
- **SRSWR** (Simple random sample with replacement of size s): this sampling is similar to SRSWOR, except that when a sample is drawn, it can be redrawn again, potentially.
- **Stratified sampling**: If D is divided into mutually disjoint parts called strata, a stratified sample of D is generated by obtaining a sample at each stratum. This helps ensure a representative sample, each stratum need to be defined manually.

Itemset mining

3.1 Definitions

Definition 3.1. An itemset is a set of items and an itemset containing k items is called a k-itemset.

Frequent itemset mining finds associations and correlations between items in large transactional or relational datasets. It is widely used in market basket analysis for example. It is the analysis of the buying habits of customers by finding associations between the different items that customers place in their shopping basket.

Definition 3.2. Itemset Y is called a **superitemset** of X if X is a **subitemset** of Y, i.e. if $X \subset Y$. Every items of X is in Y but at least one item in Y does not appear in X.

Definition 3.3. An itemset X is **closed** in $\mathcal{D}$ if there exists no proper superitemset Y such that Y has the same support count as X in $\mathcal{D}$.

Definition 3.4. A transactional dataset is a collection of transactions, where each transaction is a set of items.

With $\mathcal{L}$ the set of possible items, $\mathcal{T}$ the set of possible transactions, a transactional dataset $\mathcal{D}$ can be seen as the function $\mathcal{D} : \mathcal{T} \to 2^{\mathcal{L}}$. It can also be represented as a binary matrix, where rows represent transactions and columns represent items, mathematically it is like the function $\mathcal{D} : \mathcal{T} \times \mathcal{L} \to \{0, 1\}$.

Definition 3.5. An itemset $l \subset \mathcal{L}$ covers (or matches) a transaction $t \subset \mathcal{T}$ if every item from l is in t. Consider the match function, with $D(t, l)$ the function representing the transactional dataset:

$$\texttt{match}(l, t) = \begin{cases} 1 & \text{if } \forall i \in l, D(t, i) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Definition 3.6. The occurrence frequency of an itemset (also called frequency, support count, count or absolute support) is the number of transactions that contain the itemset (i.e., the size of the cover)

$$\texttt{support}_{\text{count}}(l) = \sum_{t \in \mathcal{T}} \texttt{match}(l, t) \tag{3.2}$$

An itemset is called frequent if its support count is greater than a minimum support count threshold θ. If an itemset is frequent then all of its subsets are also frequent.

Definition 3.7. The relative support of an itemset l in a database $\mathcal{D}$ is the percentage of transactions containing the itemset:

$$\texttt{support}_{\mathcal{D}}(l) = \frac{\texttt{support}_{\texttt{count}}(l)}{|\mathcal{T}|} \tag{3.3}$$

Definition 3.8. An itemset X is a **maximal frequent itemset** in $\mathcal{D}$ if X is frequent, and there exists no superitemset Y such that Y is frequent in $\mathcal{D}$.

Definition 3.9. An association rule represents an implication of the form $X \Rightarrow Y$, where X and Y are itemsets.

Definition 3.10. The rule support is the support of the itemset containing all the elements of the rule $\texttt{support}_{\texttt{count}}(X \Rightarrow Y) = \texttt{support}_{\texttt{count}}(X \cup Y)$.

We consider an association rule or an itemset as frequent if its support is greater than a user-defined minimum support threshold θ.

Definition 3.11. The rule confidence is the percentage of transactions containing X that contains Y too

$$\texttt{confidence}(X \Rightarrow Y) = P(Y|X) = \frac{\texttt{support}_{\texttt{count}}(X \cup Y)}{\texttt{support}_{\texttt{count}}(X)} \tag{3.4}$$

3.2 Apriori algorithm

The key idea behind the Apriori algorithm is that all nonempty subsets of a frequent itemset must also be frequent.

Algorithm 1 Apriori algorithm

Input: Transactional dataset $\mathcal{D}$, minimum support threshold θ
Output: Set F of frequent itemsets
$F = \emptyset$
$C_1 = \{\{i\} : i \in \mathcal{L}\}$
$L_1 = \{c \in C_1 : \texttt{support}_{\texttt{count}}(c) \geq \theta\}$
$F = F \cup L_1$
$k = 2$
while L_k is not empty **do**
    $C_k = \texttt{apriori\_gen}(L_{k-1})$
    $L_k = \{c \in C_k : \texttt{support}_{\texttt{count}}(c) \geq \theta\}$
    $F = F \cup L_k$
    $k = k + 1$
end while
return F

The `apriori_gen`(L_{k-1}) function generates candidate k-itemsets from the frequent (k-1)-itemsets L_{k-1} by joining them and pruning those that have infrequent subsets.

Algorithm 2 apriori_gen(L_{k-1})

Input: Set L_{k-1} of frequent (k-1)-itemsets
Output: Set C_k of candidate k-itemsets
$C_k = \emptyset$
for each $l_1 \in L_{k-1}$ **do**
    for each $l_2 \in L_{k-1}$ **do**
        if $l_1[:k-2] = l_2[:k-2]$ and $l_1[k-1] < l_2[k-1]$ **then**
            $c = l_1 \cup l_2$
            if all (k-1)-subsets of c are in L_{k-1} **then**
                $C_k = C_k \cup \{c\}$
            end if
        end if
    end for
end for
return C_k

The number of candidate itemsets is bounded by $\mathcal{O}\left(\binom{|\mathcal{L}|}{k}\right)$.
The two mains weaknesses of the Apriori algorithm are:

- Potential generation of huge candidate sets
- Repeated scan of the database and checks for pattern matching to the transactions

The Apriori algorithm can be improved using:

- **Hash-based techniques**: Instead of building L_2 from the join operation, build it similarly to L_1. When scanning the transactions, generate the 2-itemset subset of the transaction, hash them and map them to corresponding buckets containings counters to be increased. L_2 is the set of the 2-itemsets associated to buckets of a count higher than θ
- **Transaction reduction**: A transaction that does not contains any frequent k-itemsets cannot contain any frequent $(k+1)$-itemsets. They can be flagged and avoided at next steps.
- **Partitioning**: The dataset can be partitioned in N non-overlapping sub-database. Given θ, the minimum support count threshold for the whole database, $\theta' = \lfloor \frac{\theta}{N} \rfloor$ is the threshold for each partition. The candidate set C_k is the union of all local frequent k-itemset of each partition as any itemset that is potentially frequent with respect to D must occur as a frequent itemset in at least one of the partitions.
- **Sampling**: The frequent itemsets of a sample of the database are computed to serve as C_k. Some degree of accuracy is traded for eciency, thus a lower value than θ' is used to counterbalanced the effect.

3.3 FP-growth algorithm

For the FP-growth algorithm, we first need to create the FP-tree, which is a tree structure that represents all of the transactions in the database.

Algorithm 3 `fp_growth`($\mathcal{D}$, θ)

Input: Transactional dataset $\mathcal{D}$, minimum support threshold θ
Output: Set F of frequent itemsets
tree = `fp_tree`($\mathcal{D}$)
F = `fp_growth`(tree, θ)
return F

First we need to build the FP-tree, which is done like this:

Algorithm 4 `fp_tree`($\mathcal{D}$, θ)

Input: Transactional dataset $\mathcal{D}$, minimum support threshold θ
Output: A FP-tree
Scan D and compute the support count of each item, and keep only those with support count $\geq \theta$ in F
Sort F in descending order of support count, creating L the list of frequent items
Create the root of the tree
for each transaction $\tau \in \mathcal{D}$ **do**
    Sort the items of τ following L
    Add the transaction τ to the tree, by incrementing counters along the way and creating new nodes is the path is not present
end for
return Tree

Then we can mine the FP-tree to find all the frequent itemsets. The mining algorithm, works like this:

Algorithm 5 `fp_growth_mining`(Tree, α)

Input: The FP-tree and a suffix α (initially empty)
Output: Set F of frequent itemsets
if Tree contains a single path P **then**
    for each combination (denoted as β) of the nodes in the path P **do**
        generate pattern $\beta \cup \alpha$ with $\texttt{support}_{\texttt{count}}$ = min($\texttt{support}_{\texttt{count}}$ of nodes $\in \beta$)
    end for
else
    for each a_i in the header of Tree **do**
        generate pattern $\beta = a_i \cup \alpha$ with $\texttt{support}_{\texttt{count}} = \texttt{support}_{\texttt{count}}(a_i)$
        construct β's conditional pattern base and then β's conditional FP-tree $Tree_\beta$
        if $Tree_\beta \neq \emptyset$ **then**
            `fp_growth_mining`($Tree_\beta$, β)
        end if
    end for
end if

And after mining the FP-tree, we will get all the frequent itemsets.

3.4 Eclat algorithm

This algorithm works with a vertical data format, it means that we works with sets of transactions for each item, instead of sets of items for each transaction. The key idea is to use the intersection of transaction sets to compute the support count of itemsets. We can write the algorithm like this:

Algorithm 6 `eclat_algo`($\mathcal{D}, \theta$)

Input: Transactional dataset $\mathcal{D}$, minimum support threshold θ
Output: Set F of frequent itemsets
Transform $\mathcal{D}$ into a vertical data format
$F = \emptyset$
$F \cup \{i : \text{if size}(t(i)) \geq \theta\}$ for all item transaction sets $t(i)$ of item i
$C = F$
while C is not empty **do**
    $new\_C = \emptyset$
    for each pair of transaction sets A and B in C **do**
        c = $A \cap B$
        if size$(c) \geq \theta$ **then**
            $new\_C \cup c$
        end if
    end for
    $F = F \cup new\_C$
    $C = new\_C$
end while
return F

3.5 Association rule mining

Algorithm 7 `Association_rule_mining_algo`$(\mathcal{D}, \theta, \epsilon)$

Input: Transactional dataset $\mathcal{D}$, minimum support threshold θ, minimum confidence threshold ϵ
Output: Set FC of frequent and confident itemsets
$F =$ `apriori_algo`$(\mathcal{D}, \theta)$ (OR other frequent itemset mining algorithm)
$FC = \emptyset$
Generate set S the powerset ($\backslash\{\emptyset\}$) of F (i.e., all non empty subsets of F)
for each itemset $s \in S$ **do**
    Compute confidence of the rule $s \rightarrow (F \backslash s)$ using the support counts of s and F:

$$\texttt{confidence}(s \rightarrow (F \backslash s)) = \frac{\texttt{support}_{\texttt{count}}(F)}{\texttt{support}_{\texttt{count}}(s)} \tag{3.5}$$

    if confidence $\geq \epsilon$ **then**
        $FC = FC \cup \{s\}$
    end if
end for
return FC

3.6 Correlation measures

Even when we observe strong associations rules between items, they might not be interesting. To decide that, we can either manually chose if the pattern is interesting or not, it is called **Subjective assessement**. Or we can use **Objective assessment** which is based on statistics. For that, we can introduces the following measures:

Definition 3.12. The lift of a pair of item analyses the dependence and the correlation between them. It is defined as:

$$\texttt{lift}(A, B) = \frac{P(A \cup B)}{P(A)P(B)} = \frac{P(B|A)}{P(B)} = \frac{conf(A \Rightarrow B)}{P(B)} \tag{3.6}$$

- If $\texttt{lift}(A, B) < 1$, A and B are negatively correlated, i.e., the occurence of one likely leads to the absence of the other
- If $\texttt{lift}(A, B) = 1$, A and B are independant
- If $\texttt{lift}(A, B) > 1$, A and B are positively correlated, i.e., the occurence of one implies the occurence of the other

Definition 3.13. Based on the contingency table of the two items, we can compute the χ^2 **(chi-square)** statistics that tests the independence of the two items. The test is based on statistics table, and degrees of freedom. The degrees of freedom can be computed with:

$$DF = (r - 1)(c - 1) \tag{3.7}$$

And the χ^2 metrics is computed with:

$$\chi^2 = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(o_{ij} - e_{ij})^2}{e_{ij}} \tag{3.8}$$

with o_{ij} the observed frequency of the event (A_i, B_j) and e_{ij} the expected frequency of the event (A_i, B_j). e_{ij} is computed as:

$$e_{i,j} = \frac{\texttt{count}(A = a_i) \times \texttt{count}(B = b_j)}{Total} \tag{3.9}$$

with n the number of data tuples.

Example of a contingency table for two items A and B:

$o_{i,j}$	A	$\neg A$	Total
B	4000	3500	7500
$\neg B$	2000	500	2500
Total	6000	4000	10000

And the expected table is:

$e_{i,j}$	A	$\neg A$
B	4500	3000
$\neg B$	1500	1000

And here is an example of the χ^2, for different confidence values of the pair (A, B):

	P										
DF	**0.995**	**0.975**	**0.20**	**0.10**	**0.05**	**0.025**	**0.02**	**0.01**	**0.005**	**0.002**	**0.001**
1	0.0000393	0.000982	1.642	2.706	3.841	5.024	5.412	6.635	7.879	9.550	10.828
2	0.0100	0.0506	3.219	4.605	5.991	7.378	7.824	9.210	10.597	12.429	13.816
3	0.0717	0.216	4.642	6.251	7.815	9.348	9.837	11.345	12.838	14.796	16.266
4	0.207	0.484	5.989	7.779	9.488	11.143	11.668	13.277	14.860	16.924	18.467

Figure 3.1: Example of χ^2 table

Definition 3.14. All confidence measure is the minimum confidence of the two association rules related to A and B ($A \Rightarrow B$ and $B \Rightarrow A$). And it is defined as:

$$\begin{aligned} \texttt{all\_conf}(A, B) &= \frac{\texttt{support}_{\texttt{count}}(A \cup B)}{\max\{\texttt{support}_{\texttt{count}}(A), \texttt{support}_{\texttt{count}}(B)\}} \\ &= \min\{P(B|A), P(A|B)\} \end{aligned} \tag{3.10}$$

Definition 3.15. Max confidence measure is also the maximum confidence of the two association rules related to A and B ($A \Rightarrow B$ and $B \Rightarrow A$). And it is defined as:

$$\texttt{max\_conf}(A, B) = \max\{P(B|A), P(A|B)\} \tag{3.11}$$

Definition 3.16. Kulczynski measure can be view as an average confidence of the two association rules related to A and B ($A \Rightarrow B$ and $B \Rightarrow A$). And it is defined as:

$$\texttt{kulc}(A, B) = \frac{1}{2}(P(A|B) + P(B|A)) \tag{3.12}$$

Definition 3.17. Cosine measure can be view as a harmonized lift measure and is defined as:

$$\begin{aligned} \texttt{cosine}(A,B) &= \frac{\texttt{support}_{\text{count}}(A \cup B)}{\sqrt{\texttt{support}_{\text{count}}(A)\texttt{support}_{\text{count}}(B)}} \\ &= \frac{P(A \cup B)}{\sqrt{P(A)P(B)}} = \sqrt{P(A|B)p(B|A)} \end{aligned} \tag{3.13}$$

For all thoses measures, from the all confidence to the cosine measure, the measures are in the range $[0,1]$ and they can be interpreted as:

- If $\texttt{cosine}(A,B) < 0.5$, A and B are negatively correlated
- If $\texttt{cosine}(A,B) = 0.5$, A and B are independant
- If $\texttt{cosine}(A,B) > 0.5$, A and B are positively correlated

To decide whether two items are correlated or not, we can use all the previous mesures. We can also check the relation between two items by testing their conditional probabilities (i.e. $P(A|B)$ and $P(B|A)$). And with that, we can introduce the imbalance ratio.

Definition 3.18. The imbalance ratio measures the difference between the two conditional probabilities and return a value near 0 if the two probabilities are close and otherwise the closer it is to 1, the more imbalanced the two probabilities are. It is defined as:

$$\texttt{IR(A,B)} = \frac{|\texttt{support}_{\text{count}}(A) - \texttt{support}_{\text{count}}(B)|}{\texttt{support}_{\text{count}}(A) + \texttt{support}_{\text{count}}(B) - \texttt{support}_{\text{count}}(A \cup B)} \tag{3.14}$$

Usually it is advised to use kulc and the imbalance ratio together to decide.

3.7 Multilevel associations and mining rare patterns

3.7.1 Multilevel associations mining

Items can have multiple level of abstraction. For example, a store can sell different type of goods (fruits, vegetables, drinks, etc) and in those categories, there can be different subcategories (for drink: alcool, softs, milks, etc) and in those subcategories, we can have different items. Patterns can be mined at each level of abstraction, and usually patterns at more detailled levels are the most interesting. From the levels of abstraction, we can draw a tree (which is not unique), to show a way of categorizing the items. For example, we can have this concept hierarchy:

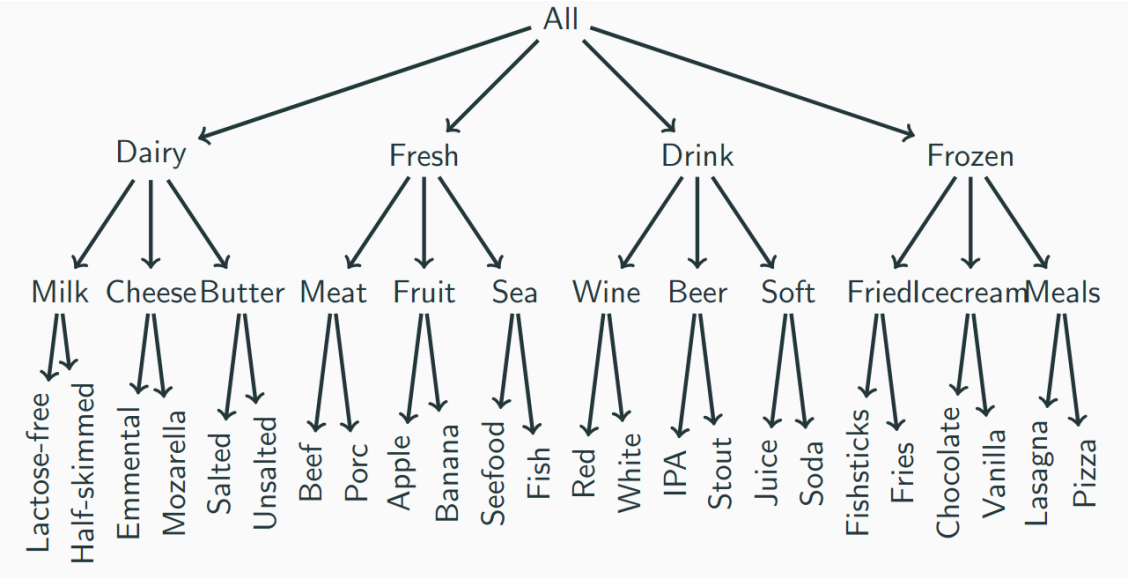


Figure 3.2: Example of a concept hierarchy

We can mine multilevel association rules by using 3 differents types of support thresholds:

Definition 3.19. Uniform support mining uses the same minimum support threshold for all the levels of the hierarchy. With this type of support, we do not consider an itemset without a frequent ancestor.

Definition 3.20. Reduced support mining uses a lower minimum support threshold for the lower levels of the hierarchy. And so with this, each level as it's own minimum support threshold, and an item might be frequent even though its ancestor is not frequent.

Definition 3.21. Group-based support mining uses specialized threshold for each group of items. For example, low price value items have a higher threshold, high price value items have a lower threshold. Usually, the mining is done with the lower support, followed by post-processing step to remove itemset that do not meet their own categories threshold

3.7.2 Mining rare patterns

Rare patterns or patterns that reflect a negative correlation between items are sometimes interesting to find.

Definition 3.22. A **negatively correlated pattern** is a pattern $A \cup B$, such that A and B are frequent but they rarely occur together, which means that:

$$\texttt{support}_{\texttt{count}}(A \cup B) < \texttt{support}_{\texttt{count}}(A)\texttt{support}_{\texttt{count}}(B) \tag{3.15}$$

And thus, A and B are negatively correlated.

Definition 3.23. A **strongly negatively correlated pattern** is a pattern $A \cup B$, such that A and B are frequent but they rarely occur together, which means that:

$$\texttt{support}_{\texttt{count}}(A \cup B) \ll \texttt{support}_{\texttt{count}}(A)\texttt{support}_{\texttt{count}}(B) \tag{3.16}$$

And thus, A and B are strongly negatively correlated.

But the formula is not null invariant, another way to check if two items are strongly negatively correlated is:

$$\begin{aligned}\texttt{support}_{\text{count}}(A \cup \neg B)\texttt{support}_{\text{count}}(\neg A \cup B) \gg \\ \texttt{support}_{\text{count}}(A \cup B)\texttt{support}_{\text{count}}(\neg A \cup \neg B)\end{aligned} \tag{3.17}$$

But is also not null invariant. To really have a null invariant measure, we can use the Kulczynski measure:

$$\frac{1}{2}(P(A|B) + P(B|A)) < \delta \tag{3.18}$$

with δ a user-defined negative pattern threshold.

3.8 Constraint-based pattern mining

In addition to the suppport, it can be interesting to find patterns that satisfies some constraints. For example, we can be interested in finding patterns that contains a specific item, or patterns with a total value under a specific budget (for a supermarket for example). There exists different types of constraints:

- **Knowledge** type constraints: specify the type of knowledge to be mined (e.g., association, correlation, classification, clustering,...)
- **Data** constraints: set of task-relevant data (e.g., sequence mining, graph mining,...)
- **Dimension/level** constraints: specify the desired dimensions of the data, the abstraction levels, the level of the concept hierarchies
- **Interestingness** constraints: specify the thresholds on statistical measures (e.g., support, confidence, correlation)
- **Rule/pattern** constraints: specify the form of, or conditions on, the rules/patterns to be mined (e.g., max/min number of items, templates, relations between attributes)

Thoses different types of constraints can be categorized in 3 different categories with different properties.

Definition 3.24. A constraint C is **pattern antimonotonic**, if an itemset does not satisfy constraint C, none of its supersets will satisfy C (e.g., the support constraint)

When a constraint is pattern antimonotonic, we can prune the search space by not considering the supersets of an itemset that does not satisfy the constraint.

Definition 3.25. A constraint C is **pattern monotonic** if an itemset satisfy constraint C, all of its supersets will satisfy C.

In opposition to pattern antimonotonic constraints, when a constraint is pattern monotonic, we cannot prune the search space, it is only a constraint that is useful to filter the results at the end of the mining process. It can also help to reduces the number of querries done to check if a superset satisfies the constraint because we only need to check if the superset contains an accepted set.

Definition 3.26. Convertible constraint is a constraint that can be transformed into a pattern monotonic or pattern antimonotonic constraint by reordering the items in the itemsets.

Definition 3.27. A constraint c is **succinct** if it can be enforced by directly pruning some data objects from the database (**data succinct**) or by directly enumerating all and only those sets that are guaranteed to satisfy the constraint (**pattern succinct**).

3.8.1 Common constraints and their categories

Constraints	Antimonotonic	Monotonic	Succinct
$v \in S$		$\surd$	$\surd$
$S \supseteq V$		$\surd$	$\surd$
$S \subseteq V$	$\surd$		$\surd$
$\min(S) \leq v$		$\surd$	$\surd$
$\min(S) \geq v$	$\surd$		$\surd$
$\max(S) \leq v$	$\surd$		$\surd$
$\max(S) \geq v$		$\surd$	$\surd$
$\texttt{count}(S) \leq v$	$\surd$		
$\texttt{count}(S) \geq v$		$\surd$	
$\texttt{sum}(S) \leq v (\forall a \in S, a \geq 0)$	$\surd$		
$\texttt{sum}(S) \geq v (\forall a \in S, a \geq 0)$		$\surd$	
$\texttt{range}(S) \leq v$	$\surd$		
$\texttt{range}(S) \geq v$		$\surd$	
$\texttt{avg}(S) \leq v$	convertible	convertible	
$\texttt{avg}(S) \geq v$	convertible	convertible	
$\texttt{support}(S) \leq \theta$	$\surd$		
$\texttt{support}(S) \geq \theta$		$\surd$	
$\texttt{range}(S) \leq \epsilon$	$\surd$		
$\texttt{range}(S) \geq \epsilon$		$\surd$	

3.8.2 constraint programming approach

Constraint programming is a powerful paradigm for solving combinatorial problems, including constraint-based pattern mining. It allows us to express complex constraints and search for solutions that satisfy those constraints efficiently. In the context of constraint-based pattern mining, we are going to use 1 boolean variables for each item and transaction. And then we are going to link those variables with constraints. And we can find all the patterns that satisfy the constraints by exploring the search space of the variables and checking the constraints.
So we are going to define the variables for the items as X_i and for the transactions as Y_i. Let's define the basic constraints:

- **Cover constraint**: if we select a transaction then we need at least to select all items in that transaction.

$$Y_m = 1 \Leftrightarrow \sum_k X_k \times \mathcal{D}_{\tau_m, i_k} = \sum_k X_k \tag{3.19}$$

- **Minimum support constraint**: we need to select at least θ transactions.

$$\sum_m Y_m \geq \theta \tag{3.20}$$

With those, we can find all the frequent itemsets. But we can also add other constraints to find more specific patterns:

- **Sum of values constraint**: we want that the sum of the values of the items selected respects a budget. (works also with minimum value of the sum)

$$\sum_k X_k \times v_k \leq P \tag{3.21}$$

- **Maximum values constraint**: we want that the maximum value of the items selected is under a specific threshold.

$$\max_k X_k \times v_k \leq P \tag{3.22}$$

And finally, if we want to find patterns that are not only frequent but also maximal or closed, we can use the following constraints:

- **Maximal patterns constraint**:

$$X_k = 1 \Leftrightarrow \sum_m Y_m \times \mathcal{D}_{\tau_m, i_k} \geq \theta \tag{3.23}$$

- **Closed patterns constraint**:

$$X_k = 1 \Leftrightarrow \sum_m Y_m \times \mathcal{D}_{\tau_m, i_k} = \sum_m Y_m \tag{3.24}$$