**Implementation of mining implication bases from knowledge graphs using FCA**

Formal Concept Analysis (FCA) is a mathematical framework for data analysis and knowledge representation. It is often used for knowledge discovery and data mining in large datasets, including knowledge graphs. FCA is particularly useful for extracting association rules or implication rules from datasets, which can help to identify patterns and relationships between data elements. In this context, an implication base is a set of implication rules that can be derived from a knowledge graph using FCA.

To implement mining implication bases from knowledge graphs using FCA, the following steps can be taken:

1. Constructing a formal context:

We take the example:

ex:a rdf:type ex:Female .

ex:a ex:sibling ex:b .

ex:b rdf:type ex:Female .

ex:b ex:sibling ex:c .

ex:b ex:child ex:d .

ex:c ex:child ex:e .

To construct the formal context for this example

Identify the entities: The entities in this example are ex:a, ex:b, ex:c, ex:d, and ex:e.

Identify the relationships: The relationships in this example are rdf:type, ex:sibling, and ex:child.

Creating a binary matrix:

Table

Description automatically generated

In this matrix, a 1 is placed in the cell if an entity has the corresponding relationship with another entity. For example, ex:a has a sibling relationship with ex:b, so there is a 1 in the cell for ex:a and ex:sibling.

This is the formal context for the knowledge graph in the example. We can use FCA to analyze this matrix and identify concepts and implication rules that represent patterns and relationships in the graph.

1. Compute the Concept Lattice:

Using the matrix we constructed earlier, we can compute the concept lattice using the Next-Closure algorithm.

Step 1: Initialize the set of closed sets to {∅}

Step 2: For each object and attribute in the matrix, compute its intent and add it to the set of closed sets if it is not already in the set.

Step 3: For each closed set in the set of closed sets, compute its extent and add it to the set of closed sets if it is not already in the set.

Step 4: Repeat steps 2 and 3 until no new closed sets can be added to the set of closed sets.

The resulting concept lattice is as follows:

{ex:a, ex:b, ex:c, ex:d, ex:e}

/ \

{ex:a, ex:b, ex:c, ex:d} {ex:b, ex:c, ex:e}

| |

{ex:a, ex:b, ex:c} {ex:b, ex:c}

| |

{ex:a, ex:b} {ex:b}

| |

{ex:a} { }

In this lattice, each node represents a closed set of objects (entities) and each edge represents an inclusion relation between concepts (sets). The top node represents the concept that includes all objects, and the bottom nodes represent the concepts that include only one object. For example, the node {ex:b} represents the concept that includes only the object ex:b.

We can interpret the lattice as a hierarchy of concepts, where the more general concepts are at the top and the more specific concepts are at the bottom. The top concept is the empty set, which represents the most general concept that includes all objects in the context. The bottom concepts represent the most specific concepts that include only a single object.

1. Generate Implication rules:

Using the concept lattice that we computed earlier, we can generate implication rules by looking at each pair of adjacent concepts (sets) in the lattice. An implication rule is of the form A → B, where A and B are sets of attributes (predicates) in the knowledge graph, and A → B holds if for all objects that have all attributes in A, they also have all attributes in B.

For each pair of adjacent concepts, we can generate an implication rule of the form A → B, where A is the set of attributes in the parent concept (set) and B is the set of attributes in the child concept (set). The implication rule holds because all objects that have all attributes in the parent concept (set) also have all attributes in the child concept (set).

Here are the implication rules that we can generate from the concept lattice:

1. { } → {ex:a, ex:b, ex:c, ex:d, ex:e}

2. {ex:a, ex:b} → {ex:c}

3. {ex:a, ex:b} → {ex:d}

4. {ex:a, ex:b, ex:c} → {ex:d}

5. {ex:b} → {ex:c, ex:e}

6. {ex:b, ex:c} → {ex:e}

7. {ex:b, ex:e} → {ex:c}

Rule 1 is a trivial rule that holds for all sets of attributes, as all objects have all attributes in the empty set.

Rules 2 and 3 state that if an object has attributes ex:a and ex:b, then it also has attribute ex:c or ex:d. These rules capture the sibling relationship between ex:c and ex:d.

Rule 4 states that if an object has attributes ex:a, ex:b, and ex:c, then it also has attribute ex:d. This rule captures the fact that ex:c is a child of ex:b and ex:d is a child of ex:b and ex:c.

Rules 5, 6, and 7 capture the relationships between ex:b, ex:c, and ex:e. Rule 5 states that if an object has attribute ex:b, then it also has attributes ex:c and ex:e. Rules 6 and 7 capture the fact that ex:c and ex:e are both children of ex:b, and that they are mutually exclusive, i.e., an object cannot have both attributes at the same time.

1. Prune Redundant rules:

After generating the implication rules, we need to prune the redundant rules, i.e., the rules that can be inferred from other rules.

We can see that rule 1 is not redundant, as it is the most general rule that holds for all sets of attributes. However, rules 2, 3, and 4 are redundant, as rule 4 subsumes rules 2 and 3. In other words, if an object has attributes ex:a, ex:b, and ex:c, it also has attributes ex:c and ex:d.

Similarly, rules 6 and 7 are redundant, as they can be subsumed by rule 5. In other words, if an object has attribute ex:b and ex:c (rule 6), then it also has attribute ex:e (rule 5), and if an object has attribute ex:b and ex:e (rule 7), then it also has attribute ex:c (rule 5).

Therefore, the non-redundant implication rules are:

1. { } → {ex:a, ex:b, ex:c, ex:d, ex:e}

4. {ex:a, ex:b, ex:c} → {ex:d}

5. {ex:b} → {ex:c, ex:e}

These rules capture the essential patterns and relationships in the knowledge graph without redundancies.

1. Evaluate and Interpret Results

The formal concept analysis and implication rule generation on the given knowledge graph produced interesting results. The concept lattice that we computed shows us the subsumption relationships between the sets of attributes in the knowledge graph. From the lattice, we can see that the most general set of attributes (the top node in the lattice) includes all the attributes in the knowledge graph. The lattice also shows us the concept hierarchy, where each child concept (set) is more specific than its parent concept (set).

The implication rules that we generated from the concept lattice capture the patterns and relationships in the knowledge graph. Rule 1 states that all objects in the knowledge graph have all the attributes in the graph, which is a trivial rule. Rule 4 captures the fact that ex:c is a child of ex:b and ex:d is a child of ex:b and ex:c. Rule 5 captures the relationships between ex:b, ex:c, and ex:e, where an object with attribute ex:b also has either ex:c or ex:e or both.

By pruning the redundant rules, we obtained a smaller set of implication rules that still capture the essential relationships in the knowledge graph. The non-redundant rules show us that if an object has attributes ex:a, ex:b, and ex:c, it also has attribute ex:d. Rule 5 shows us the relationship between ex:b, ex:c, and ex:e, where an object with attribute ex:b also has either ex:c or ex:e or both.

Overall, the formal concept analysis and implication rule generation allowed us to discover and summarize the patterns and relationships in the knowledge graph in a more structured and formal way. These results can help us understand the knowledge graph better and potentially derive new insights or knowledge from it.