XQuake as a Constraint-Based Mining Language

Valerio Grossi, Andrea Romei

CoCoMile 2012
First workshop on COmbining COnstraint solving with MIining and LEarning
Motivation and objective

- The amount of information coding XML data is growing
  - Systems for storing and querying XML data exist
  - Systems supporting DM features out of XML data are still missing

- Our goal is to mine XML data according to the principles of the inductive databases theory (IDBs)
  - We give the main intuition that is behind a constraint-based mining language out of XML data
XQuake at a glance

XQuery

- Applications can use XQuery for simple data manipulation/querying or for control structures

XQuake

- XQuake is a language/system that extends XQuery with data mining features

Raw data DM models

- According to the IDB, data and mining models are stored in a native XML DB
- Data mining is performed where the data is stored (i.e. no data transformation/manipulation)

Native XML DB
XML-based vs. Relational-based

Native XML DB
- XQuery
- Raw data DM models
- XQuake

Relational DB
- SQL
- Mining views, Atlas, DMQL, MineRule, ...
- Raw data DM models

...
Mining constructs

- XQuake admits several operators for pre-processing, mining and post-processing
- Each mining operator is made up of a combination of base constructs.
  - The syntax is an adaptation of the XQuery syntax
  - The output result is always an XML sequence
- Base constructs include:
  - Data and models iterators
  - Data/model binder
  - Constraints specification
  - Output constructor
## Mining constructs (2)

<table>
<thead>
<tr>
<th></th>
<th>Data Iterator</th>
<th>Model Iterator</th>
<th>Data/model binder</th>
<th>Constraints</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preproc.</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Mining</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Model application</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Model evaluation</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Filtering</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Main idea

- XQuake supports only simple constraints
  - E.g. «extract association rules having two items in the antecedent and the item ‘bread’ in the consequent»
- We aim at integrating domain-specific constraints
  - How to represent the background knowledge?
  - How to express the constraint?
  - How to maintain the closure principle?
- Our solution consists in
  - Representing the background knowledge with the aid of an ontology (RDF/OWL)
  - Expressing constraints directly via XQuery predicates
    - A built-in function library is used to query the ontology
Main idea (2)

- The result is in an integrated environment in which all mining entities are represented via XML.
A simple example of use

- A domain expert investigates for a future promotional campaign during the holidays (MBA)
  - The goal is to study the relation between the most frequent drinks promoted at Easter, and the most frequent cakes promoted at Christmas in the past.

Input data: XML transactions

Domain knowledge: OWL document
A simple example of use (2)

- We aim at extracting association rules having the following form:

\[(i_1 \in \text{EasterDrink}) \text{ and } (i_2 \in \text{AnyItem}) \text{ and } \ldots \text{ and } (i_n \in \text{AnyItem}) \Rightarrow (i_{n+1} \in \text{ChristmasCake}) \text{ [supp]} [\text{conf}]\]

- Where:
  - EasterDrink (resp. ChristmasCake) is the class of items that are drinks (resp. cakes) having an Easter (resp. Christmas) promotion
  - AnyItem is the entire set of distinct items
A possible implementation (1)

- XQuery is employed for querying and reasoning with OWL and RDF ontologies
- A built-in function library is used to navigate and to query the ontology

```
declar function local:hasRec($class, $prop) as xs:boolean {
   let $owl := owldoc("items.owl")
   return sw:hasSuperclass($owl, mfn:item(), $class) and sw:hasProperty($owl, mfn:item(), $prop, "hasRecurrency")
}
```
A possible implementation (2)

- An XQuake construct can be defined to extract association rules satisfying the given constraint.
  - The local:hasRec(…) function is directly used inside the mining operator.

```xml
(: Transactions, items and data constraints specific :)
1. for data $trans in doc("MBA")/store/purchase
   [@date > "01/01/2012" and @date < "01/05/2012"]
2. let group $item := $trans/item[@price * @qty > 5.0]
3. let supplementary $hasRec :=
   (<drink>{local:hasRec("Drink", "Easter")}</drink>,
    <cake>{local:hasRec("Cake", "Christmas")}</cake>)

(: Domain knowledge specification :)
4. having (some $v in mfn:supplementary-body()
   satisfies $v/drink) and
5. count(mfn:supplementary-head()) = 1 and
6. mfn:supplementary-head[1]/cake and
7. mfn:support() > 0.50 and mfn:confidence() > 0.70
8. return pmml
```
Final Remarks (1)

- **Flexibility**
  - As far as the modification of the domain knowledge
    - A built-in library is employed to traverse the ontology
  - As far as the introduction of different kinds of constraints
    - An XQuery predicate is employed to express constraints

- **Closure principle**
  - Data, mining models and the background knowledge are XML documents
  - XQuery (extended) is used to represent the KDD process
Final Remarks (2)

- Future work
  - Finalizing the implementation of the built-in XQuery library used to navigate the ontology
  - Exploiting domain-specific constraints for different kinds of models
    - E.g. clusters and sequential patterns

