Ontology extraction from MongoDB using formal concept analysis

Conference Paper · October 2017

DOI: 10.1109/ICKEA.2017.8169925

CITATIONS
0

READS
36

2 authors:

Simin Jabbari
Université de Neuchâtel
3 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Kilian Stoffel
Université de Neuchâtel
85 PUBLICATIONS **795** CITATIONS
SEE PROFILE

Ontology extraction from MongoDB using Formal Concept Analysis

Simin Jabbari
Information Management Institute
University of Neuchatel
Neuchatel, Switzerland
simin.jabbari@unine.ch

Kilian Stoffel
Information Management Institute
University of Neuchatel
Neuchatel, Switzerland
kilian.stoffel@unine.ch

***Abstract*—Using formal concept analysis, we propose a method for engineering ontology from MongoDB to effectively represent unstructured data. Our method consists of three main phases: (1) generating formal context from a MongoDB, (2) applying formal concept analysis to derive a concept lattice from that formal context, and (3) converting the obtained concept lattice to the first prototype of an ontology. We apply our method on NorthWind database and demonstrate how the proposed mapping rules can be used for learning an ontology from such database. At the end, we discuss about suggestions by which we can improve and generalize the method for more complex database examples.**

Keywords: MongoDB; Ontology learning; Formal Concept Analysis; Unstructured data.

I. INTRODUCTION

How to efficiently represent knowledge is crucial for data analysis especially when we face unstructured data. Nowadays, majority of data sources are unstructured. Having a systematic approach for organizing such data is highly appreciated. This facilitates further analysis and semantic derivations from data. Ontologies have been used for knowledge representation and implicit knowledge extraction. Although lots of efforts have been taken to engineer ontology [1] from relational data bases [2,3,4], only a few researchers have been worked on the problem of learning ontology from unstructured data [5]. In this paper we propose a method for learning ontology from a MongoDB (an example of document-based database for unstructured data) using formal concept analysis (FCA) [6]. It has been shown that FCA is a powerful method for ontology learning [7,8,9] and by building ontological concepts and representing their hierarchy [10]. It has been frequently used to identify relationship between a set of objects and a set of attributes. Although FCA is well-known for analyzing data across various domains, it can also be used as a suitable tool for learning ontology.

We start by reviewing the basic concepts in section II. In section III, we propose our method for learning ontology from MongoDB by defining some mapping rules. We then apply our method to a case study (NorthWind database). The proposed method has been proven to be correct by practice.

II. REVIEW BASIC CONCEPTS

A. *Ontology*

Ontology is a model for describing a domain of interest. It is a basis for conceptualization and serves like a backbone for knowledge representation by defining a domain consisting of different types, properties and relationships between them using a suitable ontology language such as OWL [11]. OWL stands for Web Ontology Language and can be considered as the most important application of Description Logic [12]. ontology simplifies sharing the knowledge and enables reasoning and inference on data with the aim of implicit knowledge extraction. It consists of a set of statements called Axioms. There are two important groups of axioms in ontology called assertional axioms (ABox) and terminological axioms (TBox). TBox defines the terminology of an application domain such as the set of concept definitions and relationship between them. ABox axioms represent the knowledge about individuals. It consists of the concepts that they belong to and their relationships with other individuals.

In OWL concepts are called classes and the roles are called properties. OWL properties are either object properties to describe relation between classes, or datatype properties to describe instances of classes. Note that the members of a class are called instances.

B. *Formal Concept Analysis*

A formal context is denoted by a triplet $< X; Y; I >$where I is a $|X| \times |Y|$ binary table that represents relation between a set of objects X (rows) and a set of attributes Y (columns). $I(x, y) = 1$ indicates that object x has attribute y. Assume $A \subseteq$ X and $B \subseteq Y$ denote a set of objects and attributes, respectively. Then a pair $(A; B)$ is a formal concept if and only if $A^u = B$ and $B^d = A$. Here, A^u denotes a set of those attributes that all objects in A have in common. Similarly, B^d denotes a set of those objects that all attributes in B have in common. Each formal concept is indicative of the usual notion of a real concept in the world. Thanks to FCA, one could build a hierarchy between all concepts (A, B), where between each pair of concepts (A_1, B_1) and (A_2, B_2) we may have a sub-concept, super-concept, or disjoint relation [6] for more detail).

C. MongoDB

MongoDB is one of the documented-oriented NOSQL databases [13,14]. It is a popular tool for storing unstructured data which has a flexible document data model that allows user to improve it as the requirements change. It stores data in collections made out of individual documents which can be nested in complex hierarchies but still easy to query and index. Unlike relational databases, one of the characteristics of MongoDB is that it has no pre-defined schema. MongoDB is one of the fastest-growing database which provides a rich document-oriented structure, and due to its popularity we

choose it as a database in our case study.

III. RESULTS

A. From MongoDB to concept lattice

As discussed earlier, document oriented MongoDB does not have a structure which makes it impossible for implicit knowledge extraction. However, we could make it possible by proposing a formal semi-automated approach for ontology development using FCA. Our approach is applicable in MongoDB in which there is no references between documents of different collections that causes a big issue for extracting taxonomy of concept.

1) Context formation:

The first step towards having an ontology from unstructured data using our approach is to build a formal context from unstructured data. In other words, for using FCA as the backbone of ontology development, we need to convert unstructured data into acceptable input for FCA algorithms by importing data into a table, called formal context. For generating formal context, a set of objects, attributes and relationship between them must be extracted. For this conversion our approach is the following:

- The set of objects X in the formal context is equal to the set of collection names in database. Once we extract all collection names we put them as row IDs in the formal context.
- Every filed name in each document of a collection is transformed into a table column. That is the set of attributes Y in the formal context becomes the list of all field names in MongoDB.
- To determine the binary relation between each object and attribute of the formal context we use the following rule: if collection x has field name y, then we put $I(x, y) = 1$.

2) Concept lattice generation

Once the formal context is created from MongoDB, we need to apply one of FCA algorithms to extract formal concepts, as well as the concept lattice. The latter provides a nice hierarchy between the concepts and can explain sub-concept, super-concept relationships between formal concepts. We used Nourine algorithm [15] to form all formal concepts, and the concept lattice. Nourine algorithm has been shown to be quite fast and efficient for generating concept lattice.

Fig. 1 depicts the concept lattice that is generated by Nourine algorithm, where the input to this algorithm is a formal context that is extracted from NorthWind database using the method described above. In Fig. 1 each node represents a concept labelled with its extensional part. The intentional part of each formal concept is given in Fig. 2.

B. From concept lattice to ontology

A concepts lattice represents a full hierarchy between the formal concepts. However, for deriving an ontology we may not need to consider all of these formal concepts. In other words, from the view point of ontology, there may exist some redundant information in concept lattice which we can exclude in our ontology. For instance, that extent and intent of a concept usually overlaps with those of its super-concept or sub-concept. There might be different approaches for dealing with redundancy. One of them is reduced labeling [7] by which we can remove some redundancy from concept lattice. In our approach, we take advantage of such redundancy for extracting ontological concepts, properties, object properties, and data type properties. Note that a formal concept (C_i, D_i) in FCA is defined as a pair of extent C_i and intent D_i. However, we only need the extensional part C_i of a formal concept to define ontological concept. Throughout this paper we may use C_i to denote ontological concept as well as extensional part of formal concept. We also use notation $|C_i|$ to denote the cardinality of the "extensional part" C_i of a formal concept.

In the following, we propose our mapping rules to be used for converting a concept lattice to an ontology.

Rule1: For each concept C_i in the set of formal concepts, if $|C_i| = 1$, we assign an ontological concept. In our example (see the concept lattice in Fig. 1), 11 ontological concepts will be created according to this rule. Examples includ *[products]* and *[order_details]*:

```
<owl:Class rdf:ID="products"/>
<owl:Class rdf:ID="order_details"/>
```

Rule 2: If $|C_i| = 2$ and it is "directly" connected to the most general concept (i.e., the concept of *"Thing"* C_k with $k = \arg\_max_i |C_i|$), then an object property P_{12} is created between the two concepts $[O_1]$ and $[O_2]$, where $C_i = [O_1, O_2]$. In the statement above, a "direct" connection to the concept of thing means there is no other concept C_j that is super concept of C_i. The domain and the range of the object property P_{12} is O_1 and O_2, respectively. The name of this object property is concatenation of O_1 and O_2.

In our example above, a new object property is defined between *[products]* and *[order_details]* with the name *"prodects_order_details"*. The domain and the range of this relation would be *[products]* and *[order_details]* concepts respectively. In OWL ontology, it will be described by:

```
<owl:ObjectProperty rdf:ID=" prodects_order_details">
        <rdfs:domain rdf:resource="#products"/>
        <rdfs:range rdf:resource="#order_details"/>
</owl:ObjectProperty>
```

Rule3: If the extensional part of a concept consists of two elements (i.e., if $|C_i| = 2$), and that concept is NOT directly connected to the most general concept *"thing"*, then there is a chance of inventing a new concept. For instance, in our example above the concept *[customers, suppliers]* is connected to the concept of *"thing"* via *[customers, suppliers, employees]*, and *[customers, suppliers, shippers]*. Therefore, by inventing a new concept *[cust_supp] = [customers, suppliers]*, we can interpret *[customers, suppliers, employees]* as an object property between *[cust_supp]* and *[employee]* (according to Rule2). The concept *[customers, suppliers, shippers]* is also interpreted as an object property between *[cust_supp]* and *[shippers]* (according to Rule2).

Rule4: If an ontological class is associated to a formal concept C_i with $intent = [a_1, a_2, \dots, a_n]$, then each element a_i of the intent set is considered as a DataTypeProperty of that ontological concept. For instance, in our example, the data type properties of the ontological concept *[order_details]* would be *[UnitPrice, Discount, Quantity, ProductID, OrderID]*. In OWL it is described by:

```
<owl:DatatypeProperty rdf:ID="UnitPrice">
     <rdfs:domain rdf:resource="#order_details">
     <rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>
...
```

Rule5: The hierarchy of the ontological classes will be determined by this rule. To determine whether an ontological concept is a sub-concept or super-concept of another ontological concept, we need to check if such a relation exists in the corresponding concepts in the concept lattice. In other words, for two formal concepts C_i and C_j that are also considered as ontological concepts, we have to check two conditions. First check whether $C_i < C_j$ (i.e., C_j is a super-concept C_i) and then check there is no other C_k such that $C_i < C_k < C_j$. If two conditions above are fulfilled, then we would say that C_i is a sub-concept C_j and C_j is a super-concept of C_i (in ontological sense).
For instance, according to Rule5, the most general concept *"thing"* is a super-concept of *[order_details]*, and *[order_details]* is a sub-concept of *"thing"*. In OWL, we describe such relation as:

```
<owl:Class rdf="order_details">
     <rdfs:subClassOf rdf:resource="#thing"/>
</owl:Class>
```

Rule6: Using OWL, it is also possible to define property axioms such as owl:inverseOf. Therefore, for each object property P_{ij} (with domain C_i and range C_j) a new object property P_{ji} (with domain C_j and range C_i) is created and is defined as the inverse of other object property. For example, a new object property *"order_details_products"* is created as the inverse of object property *"products_order_details"* which has been already created according to Rule2. In OWL, it is described by:

```
<owl:ObjectProperty rdf:ID="products-order_details">
      <owl:inverseOf     rdf:resource="#order_details-
      products"/>
</owl:ObjectProperty>
```

Rule7: OWL also specifies property cardinality. To do so, we calculate size of each field in all documents within a collection. For each data type property of an ontological class C_i, if at least one of the fields has a null value, we set owl:minCardinality = 0, otherwise we set owl:minCardinality = 1. The same idea can be applied to determine owl:maxCardinality. For instance, the minCardinality and maxCardinality of data type properties *UnitPrice, Discount, Quantity, ProductID, OrderID* is 1 as defined in the following OWL language:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#UnitPrice"/>
  <owl:minCardinality">1</owl:minCardinality>
  <owl:maxCardinality">1</owl:maxCardinality>
</owl:Restriction>
```

Rule8: For learning instances or individuals of created ontology, for each collection, we have to check if it is assigned to ontology class or ontology object property. Then individuals will be learned automatically by using the assigned values to each filed.

The engineered ontology for NorthWind database is depicted in Fig. 3 (by protégé). Note that by using FCA for making ontology, we do not need to check "equivalent" axiom in OWL, because all the concepts derived by FCA are disjoint. In fact, this is an inherent property of FCA that if the extents of two formal concepts are the same, their corresponding intents are also the same. Therefore, all the concepts generated by FCA (and so ontological concepts) are disjoint.

IV. DISCUSSIONS

Although our proposed method cannot fully build an ontology, it significantly reduces the cost and effort associated to ontology building from scratch. In other words, our algorithm builds a skeleton for the ontology that can be further completed to a full-ontology, if it is combined by an expert opinion who knows that context quite well. Building an ontology in general is not so easy. In fact, there is no specific rule that can be applied to build a suitable ontology from any unstructured data.
In previous section, we only discussed those mapping rules which can be applied almost to all concept lattices regardless of the size and the number of their concepts. In fact, we chose those that can be generalized. However, there might be other mapping rules that could help a lot for refining the ontology. Here we provide an example, but we emphasize that the reason we did not consider it as a rule is that we could not

generalize it. Therefore, we decided to discuss it in this section to give an insight to the reader and do not consider it as a general mapping rule.

We already saw that a formal concept in a lattice can be considered as a class, or object property (Rules 1, 2). We also mentioned that how new classes can be invented using formal concepts (Rule3). Here we would like to consider a situation where an existing concept can be deleted to avoid complexity of the ontology. In our example above, the triple concept *[employees, orders, employee_terittories]* can be interpreted as three object properties between each pair of *[employees], [employee_terittories],* and *[orders]*. However, a different approach is to eliminate *[employee_terittories]* (from the set of concepts), because

Intent([employee_terittories]) = *Intent([employee_terittories, employees])* ∪ *Intent([employee_terittories, employees, orders])*.

That means all the data properties of *[employee_terittories]* can be found in data properties of other classes. Once *[employee_terittories]* is deleted, then the old object property defined between *[employee_terittories]* and *[terittories]* is replaced by object properties between *[terittories]* and *[employees]* as well as between *[terittories]* and *[orders]*, to make sure that the connection from *[territories]* to other concepts in the ontology is not ruined because of the elimination of *[employee_terittories]*. Moreover, since *[employees]* and *[orders]* have been already connected, we do not need to keep both newly created object properties above. An expert can then help to determine which one we should delete. Our suggestion is to delete object property between *[orders]* and *[terittories]* because it is likely that *[terittories]* is in more relation with *[employees]* than *[orders]*.

In the above example, the cardinality of all formal concepts (except for the most general concept *"thing"*) is less than or equal to three. This is why we could simply identify whether each formal concept should be considered as an ontological concept or object property. In case of having concepts with more than 3 elements, we suggest (as a start) to iteratively reduce the size of concepts using new invented concepts (just like what we did in example above for building new concept *[cust_supp]*.

V. CONCLUSIONS

We proposed a method for building ontology from MangoDB using FCA. In our proposed method, the concept lattice (the outcome of FCA algorithms) can be converted to a backbone for an ill formed ontology which can be further improved by adding some domain knowledge and expertise. According to our proposed mapping rules, formal concepts in a concept lattice can be converted to ontological concepts or object properties. We may also generate new concepts or even delete some of the existing concepts by some criterion. We applied our method on NorthWind database to verify its capability in building an ontology for unstructured data.

REFERENCES

[1] Haav, Hele-Mai. "A Semi-automatic Method to Ontology Design by Using FCA." *CLA*. 2004.

[2] Li, Man, Xiao-Yong Du, and Shan Wang. "Learning ontology from relational database." Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on. Vol. 6. IEEE, 2005.

[3] Biskup, Joachim. "Achievements of relational database schema design theory revisited." International Workshop on Semantics in Databases. Springer, Berlin, Heidelberg, 1995.

[4] Chiang, Roger HL, Terence M. Barron, and Veda C. Storey. "Reverse engineering of relational databases: Extraction of an EER model from a relational database." *Data & Knowledge Engineering* 12.2 (1994): 107-142.

[5] Abbes, Hanen, Soumaya Boukettaya, and Faiez Gargouri. "Learning ontology from big data through MongoDB database." *Computer Systems and Applications (AICCSA), 2015 IEEE/ACS 12th International Conference of*. IEEE, 2015.

[6] Ganter, Bernhard, and Rudolf Wille. *Formal concept analysis: mathematical foundations*. Springer Science & Business Media, 2012.

[7] Fu, Gaihua. "FCA based ontology development for data integration." *Information processing & management* 52.5 (2016): 765-782.

[8] Baader, Franz, and Baris Sertkaya. "Applying formal concept analysis to description logics." *ICFCA*. 2004.

[9] Sertkaya, Baris. "A survey on how description logic ontologies benefit from formal concept analysis." *arXiv preprint arXiv:1107.2822* (2011).

[10] Wille, Rudolf. "Restructuring lattice theory: an approach based on hierarchies of concepts." *Ordered sets*. Springer Netherlands, 1982. 445-470

[11] Bechhofer, Sean. "OWL: Web ontology language." *Encyclopedia of Database Systems*. Springer US, 2009. 2008-2009.

[12] McGuinness, Deborah L., and Frank Van Harmelen. "OWL web ontology language overview." *W3C recommendation* 10.10 (2004): 2004.

[13] Chodorow, Kristina. MongoDB: The Definitive Guide: Powerful and Scalable Data Storage. " O'Reilly Media, Inc.", 2013.

[14] Han, Jing, et al. "Survey on NoSQL database." Pervasive computing and applications (ICPCA), 2011 6th international conference on. IEEE, 2011.

[15] Nourine, Lhouari, and Olivier Raynaud. "A fast algorithm for building lattices." *Information processing letters* 71.5-6 (1999): 199-204.

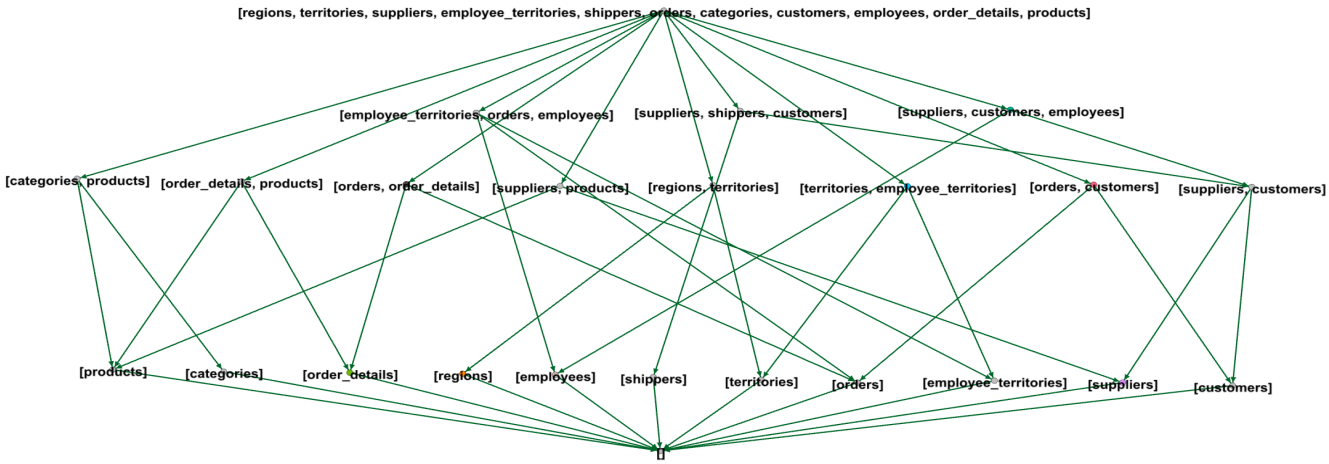

Figure 1. The concept lattice generated from NorthWind MongoDB using Nourine Algorithm

Extent	Intent
[territories]	[TerritoryID, TerritoryDescription, RegionID]
[territories, employee_territories]	[TerritoryID]
[suppliers]	[SupplierID, CompanyName, HomePage, Address, Phone, Region, Country, PostalCode, City, Fax, ContactName, ContactTitle]
[suppliers, shippers, customers]	[CompanyName, Phone]
[suppliers, products]	[SupplierID]
[suppliers, customers]	[CompanyName, Address, Phone, Region, Country, PostalCode, City, Fax, ContactName, ContactTitle]
[suppliers, customers, employees]	[Address, Region, Country, PostalCode, City]
[shippers]	[CompanyName, Phone, ShipperID]
[regions]	[RegionDescription, RegionID]
[regions, territories]	[RegionID]
[regions, territories, suppliers, employee_t...	[]
[products]	[Discontinued, CategoryID, SupplierID, UnitPrice, ProductName, QuantityPerUnit, UnitsOnOrder, ProductID, ReorderLevel, UnitsInStock]
[orders]	[ShipName, RequiredDate, ShippedDate, ShipCity, CustomerID, ShipVia, ShipPostalCode, OrderID, ShipRegion, OrderDate, field14, ShipAddress, ShipCountry, Em...
[orders, order_details]	[OrderID]
[orders, customers]	[CustomerID]
[order_details]	[UnitPrice, Discount, Quantity, ProductID, OrderID]
[order_details, products]	[UnitPrice, ProductID]
[employees]	[Address, FirstName, PostalCode, Title, Photo, City, Extension, PhotoPath, HomePhone, HireDate, ReportsTo, Region, Country, TitleOfCourtesy, LastName, Emplo...
[employee_territories]	[TerritoryID, EmployeeID]
[employee_territories, orders, employees]	[EmployeeID]
[customers]	[CompanyName, Address, Phone, Region, Country, PostalCode, CustomerID, City, Fax, ContactName, ContactTitle]
[categories]	[CategoryID, Description, Picture, CategoryName]
[categories, products]	[CategoryID]
[]	[Discontinued, ShipName, Address, ProductName, Photo, CustomerID, ShipVia, OrderID, ShipperID, UnitsInStock, SupplierID, CompanyName, Picture, ShipAddres...

Figure 2. Extents and intents of the generated formal concepts. The root of the concept lattice (in Fig. 1) is marked by dark blue color.

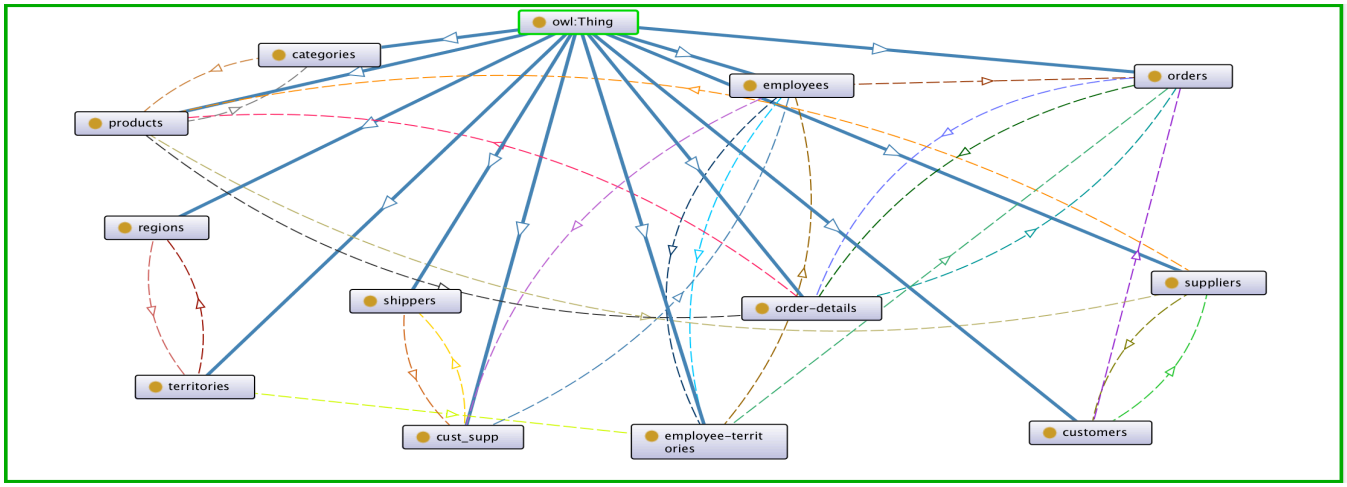

Figure 3. The ontology engineered for NorthWind database using our proposed method (in Protégé). Solid blue lines represent sub-class relationship, and the dashed lines represent object properties.