MASTER 2 DATA & KNOWLEDGE

Information Integration

Data Linking Project

Presented by :

Wafa DJERAD

Wilfried ZAKIE

2017/2018

Table of Contents

Introduction

The semantic Web have huge amount of data available in a standard format, reachable and manageable by using semantic Web tools.
In order to browse data over the Web, machines should be able to resolve the multiplicity of data references related to the same real world object, by defining correspondences among data in form of data links.

In general terms, data linking is the task of determining whether two objects descriptions refer to the same object.
This task is performed on the basis of evaluating the degree of similarity among different data instances. The higher is the similarity between two data descriptions, the higher is the probability that these descriptions actually refer to the same object.

This report presents an implementation and instructional information that describe the approach to match entities from two ontologies (ontology alignment).

I. Ontology matching approaches

The task of ontology matching can be classified into two main categories:

1. **Schema based approaches** try to infer the semantic mappings by exploiting information related to the structure of the ontologies to be matched (do not take into account the actual data classified by the ontologies).

2. **Instance based approaches** look at the information contained in the instances of each element of the schema and try to infer the relationships between the nodes of the ontologies from the analysis of their instances.

In this work, we describe the results we obtained when using some simple methods to align two ontologies, using an instance based approach.

II. General architecture

In the instance based approach, we propose techniques to compute matching scores based on the documents classified under the nodes of ontologies. The matching process starts from a pair of ontologies to be aligned. The two ontologies are traversed and the system maps their entities. It is done by computing the string similarity score between each element of the first ontology and all instances in the

second ontology. The instance of the second ontology having the best aggregate score with the instance of the first one will be the matching entity.

III. Approach :

Our approach consists of first loading the RDF files of both ontologies and store them using Jena TDB Factory library. The generated datasets are queried to obtain for each ontology every triples using a simple SparQL query:

$$?s\ ?p\ ?o$$

where s is the subject, p the relation and o is the object.

Each object related to the same subject is stored in a map structure mapping the subject to all its related objects.

Sometimes some objects may be themselves a resource, meaning a subject in some triples, for example addresses. For such objects, we get the objects related to them using another SparQL query where the subject is that object.

$$?o\ ?p\ ?o'$$

We append these objects to the subject's list of objects. Further these resources are stored in a list to avoid using them for mapping. This process help us traverse the RDF graph no matter its depth.

After each subject is mapped to all its objects for both ontologies, we will compute a similarity score between them.

Let A and B being two ontologies.

Let $E = < e_1, e_2, ..., e_m >$ and $E' = < e'_1, e'_2, ..., e'_n >$ be respectively the entities of the ontology A and B.

Each element of the ontology A will be associated to all entities of B and a corresponding score will be assigned to each pair as followed:

$$< (e_1, e'_1), score_1 >, < (e_1, e'_2), score_2 > ..., < (e_1, e'_n), score_n > ... < (e_m, e'_n), score_{m*n} >$$

The score is an aggregate score computed between all objects of the pair as followed:

Given two entities e_a, e'_b chosen randomly and belonging respectively to ontology A and B. Let $L \text{ and } L'$ being the list of the object's literals related to them such that:

$L = <l_1, l_2, ..., l_i>$ and $L' = <l'_1, l'_2, ..., l'_j>$

We will compute the match between each literal of both entities using the following algorithm:

$score = 0$
$For\ each\ l\ in\ L$
  $For\ each\ l'\ in\ L'$
    $If\ l = l'$
      $score = score + 1$
    $Else$
      $score = score + (JaroWinkler(l, l') + 2 * Jaccard(l, l'))/3$
  $end\ for$
$end\ for$

The combination of the JaroWinkler and the Jaccard methods help us to check the string spellings as well how their overlaps. We noticed giving more weight to the Jaccard score increases the precision as well as the recall. It can be interpreted as the more words they have in common the more similar they are and the JaroWinkler ensures their spelling similarity. Also this score is normalized by dividing by 3 to get values between 0 and 1.

After computing the aggregated score between entities' literals, we normalized them by dividing it by the total number of literals

$$normalizedScore = \frac{Score}{i*j} \quad \text{where}$$

We keep in a map the pairs having a weight greater than 0.8. Finally we choose as mapping of an entity e_x of A the entity e_y of B with who it has the highest score. That pair $<(e_x, e'_y)>$ is added to the output.

This process being a 1-to n mapping, we Repeat the above method reversely (i.e each entities in B will be matched to all entities in A) and combine it to the first one in order to get a $n - to - n$ mapping between these ontologies. We store the pairs $<(e_x, e'_y)>$ that are not present in the output to avoid redundancy.

IV. Experimental setting

The algorithm described in the previous section have been fully implemented in a coherent and extensible framework using the Java programming language and evaluation experiments have been run.

A. Test data:

The evaluation of ontology matching approach have been tested with “real world” data :

- Person 1
- Person 2
- And Restaurant.

B. Parameters:

1. API used:

- **Apache Jena (Jena in short):**
  is a free and open source Java framework for building semantic web and Linked Data applications. Jena is composed of different APIs interacting together to process RDF data.

- **SecondString:**
  Second String is an open-source, Java toolkit of approximate string-matching techniques. Using this API, we compared our results on matching entity names with different string distance metrics.

C. Implementation

In order to implement this algorithm, we used 5 different classes (Data, DataLinking, PreProcess, Evaluation, and Resource).

The class data creates RDF graphs using Jena TDB Factory. It contains a main method and should be run first and separately from the others. It takes as input a text file defining path of RDF files and the output path.

The other classes are linked to the *DataLinking* class which is launched with another text file where the ontologies URIs are specified as well as : gold standard path and

the output path. the output path should be the same as the one in the Data class input file.

Once Launched it passes the input file to the Preprocess data which get the information specified in. For each of these information a specific action will be taken. This class is in charge of parsing the gold standard in XML format and store it in .txt file at output folder defined above. Also, it collects the OWL URIs and store them as its class variable.

After the data files have been pre-processed, the *DataLinking* queries the RDF graphs of both ontologies to get entities to be linked. Objects of the class Resource are instantiated for all resources found while querying both graphs. The Resource's objects contain a java arraylist of literals that will store all literals linked to the resources. We then append to that list all literals found while querying. The resources that are found to be objects (in the query) of other resources are deleted after the literals linked to them are inserted to the original subject. This is done recursively using the DataLining's method *getResourceRelations().* This ensures traversal of the graphs no matter their depth.

At the end we store for each ontology in a map, the entities represented as Resource's objects and their name (URI string) as key.

After all entities have been linked to all their related literals for both ontologies, the *DataLinking* method's *mapping()* match them using the approach mentioned earlier. The output is stored in the file result.txt located inside the output folder.

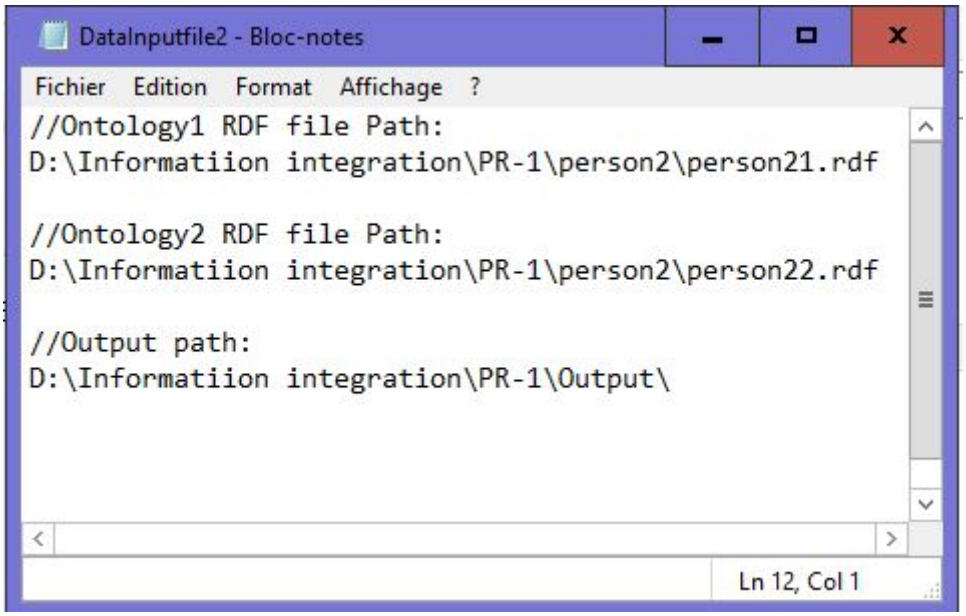
Finally the class *Evaluator* is used by *DataLinking* to compute evaluation metrics.

D. Execution

As mentioned in the implementation part, some input files should be launch along with the *Data and DataLinking .* The input files are as followed

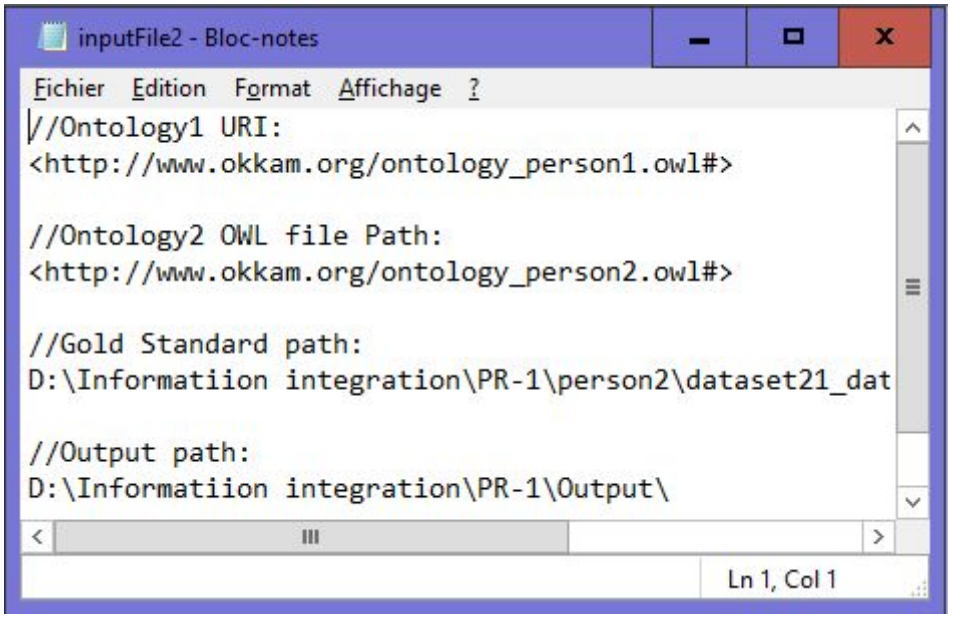
1. Data Input File

For 2 given ontologies whose entities will be mapped, the *Data* class will create RDF graph based on their RDF files. The created dataset will be stores in the output files. The file **should be presented** as in the picture below:



2. DataLinking Input File

For matching as well as evaluation the *DataLinking* class needs the OWL ontology URIs as well as the gold standard path and output folder path as described in the picture below.



Nb :

One should pay attention to the layout of the files! The path should be put at the same place as shown in the picture.

E. The evaluation:

To test the quality of our matching approach, we need to perform some evaluation metrics.

1. **Precision:**
   Also called **positive predictive value** is the fraction of relevant instance among the retrieved instances.

2. **Recall:**
   Is the fraction of relevant instance that have been retrieved over the total number of relevant instances (the gold Standard Size).

3. **F1 Score:**
   The F_1 score is the harmonic average of the precision and recall, where an F_1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

V. Results

The following results are obtained using the approach described below.

DataSets/Metrics	Person 1	Person 2	Restaurant
Precision	**99.60 %**	**82.21 %**	**77.87%**
Recall	**100.0 %**	**76.25 %**	**77.57 %**
F-Score	**99.5 %**	**79.12 %**	**77.72 %**

The Dataset 1 have a recall higher than the precision because it may have some supplementary matchings while all gold standard's matching has been found.

We get a quite accepting results for the other datasets.

VI. Conclusion

For this project we computed an algorithm for data linking based on string similarity.

This algorithm has been tested on 3 different datasets and the results have been interpreted.

Unfortunately, our approach is time and space consuming. In future work, we can overcome this by comparing entities by selecting only relevant relations for matching using SAKey API.

References :

1. https://datafireball.com/2014/10/29/jaro-winkler-string-distance/ **-** October 29, 2014 by datafireball
2. http://www.webtoolkitonline.com/xml-formatter.html - Web Toolkit Online Useful Online Tools for Developers