Semantic Data Transformation

Joffrey de Oliveira[1,2], Hammad Aslam Khan[2], Olivier Curé[1], Philippe Calvez[2]

[1] LIGM Univ Paris Est Marne la Vallée, CNRS, F-77454.
{firstname.lastname}@univ-eiffel.fr
[2] ENGIE LAB CRIGEN
philippe.calvez1@engie.com, hammadaslam.khan@engie.com

Abstract. Data transformation from diverse formats to semantic data is a major challenge that is being addressed by commercial products available in the market to some extent. However, the transformation process still requires considerable effort from users. Knowledge creation, which is one of the major steps in knowledge graph (KG) maintenance, needs existing data transformation from various formats to RDF (Resource Description Framework) data. Current data transformation approaches are either manual or through mappings. This work presents a semantic data transformation approach for knowledge creation that is semi-automatic requiring minimum possible input from users and does so with machine learning and natural language processing techniques.

Keywords: Semantic data, knowledge graph, RDF, machine learning, knowledge creation

1 Introduction

With the emergence of the Semantic Web and Linked Open Data more and more applications over the past decades have been developed to exploit semantic data. Nevertheless, the vast majority of content published on the Web is not represented in RDF and thus requires a transformation before being published. These data integration issues are very common when working with data sources on the web. There are several solutions to transform data into semantic data, most of them being based on mapping solutions which can accurately generate semantic data by writing scripts. But the writing of these mappings requires considerable effort from the share of qualified users which makes each transformation time consuming. They often require a specific mapping language and the definition of rules for each data source which makes it time consuming and prone to human error. Some solutions try to alleviate the expertise part by extending SPARQL like SPARQL Generate [2], RML [1] and R2RML [3]. In addition, there is a multitude of different data sources and each new source you want to use, demands its own transformation. This led to the development of new solutions which enable to better manage this source diversity. However, these solutions remain time-consuming on the user side since they usually imply that the user manually maps each element of their data to some ontologies. The most advanced of these

systems try to make the process semi-automatic by learning from the data and schema that match their ontologies. [6]

To ease the creation of knowledge graphs from non-semantic data, we propose an approach combining natural language processing (NLP) and machine learning to allow for a more automatized transformation of data.

2 Motivating Example

Over the past few years, more and more governments have started publishing their data for use by the public. These Open Data initiatives are done at every level of governments, from single cities to nationwide programs, and involve many different types of data. One of these is geographical data, often in the form of addresses and geographical locations of all sorts of places, such as named places, landmarks, residential buildings, or businesses. This data allows users to develop a variety of useful applications, with notable examples creating non-proprietary maps of the world such as OpenStreetMap, MapBox and others.

But currently the vast majority of data being published is not available as linked data, OpenStreetMap(OSM) created the OSMOnto [5], an ontology close to OSM's database to integrate more closely with the semantic web. Nevertheless, linked data from certain regions can remain hard to find, if it even exists. Thus the need to translate openly published relational data, into linked data. The task remains difficult as there is no single world standard describing addresses, and even when dealing with a single country there is rarely a single database regrouping all data with a single schema. It isn't unusual to find a different schema for each city when they publish the data themselves. We will attempt to explain our approach through examples obtained while transforming this type of data in Section 2 and evaluating our approach in Section 3.

3 Approach

3.1 Processing overview

Figure 1 represents our approach to data transformation. We take data sources that the user wants to map to ontologies and the description of these ontologies. We then match the columns and tables found in the data sources to the properties and classes from the ontologies. The matches are suggested based on scores from 3 different modules which are denoted NLP, String and Numeric (to be detailed in this section). But there may be cases when mapping becomes ambiguous because of lack of information. In such a scenario a mapping is proposed and the end-user is given the opportunity to verify and change some of the mappings through the graphical user interface provided with the tool we developed.

3.2 Matching classes

When a user wants to transform tabular data, the application lets her create entity types by selecting the columns she wants to see in the entity type. Here,

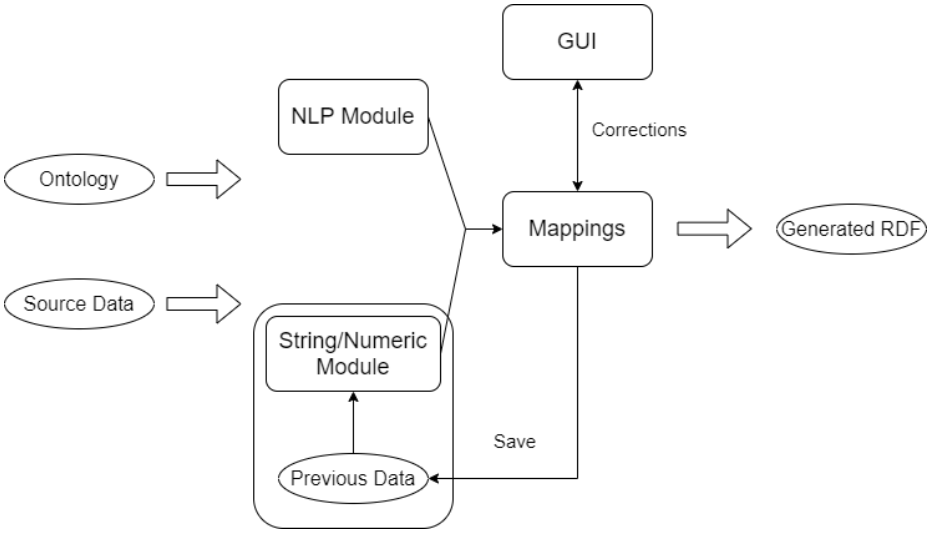

Fig. 1. Representation of our process

an entity type is a table containing a number of columns that will be matched with a class from the ontologies known by the application. The columns under the entity type will be matched to properties associated with the class. That way when generating the RDF document, each row in the entity type's table will generate an instance of the class that matched the entity type and the values in this row's column will be used to generate the properties associated with that instance. The matching is performed between properties and columns. The average score for the matching between columns from a table and properties related to a class is then used as the score for the matching between the class and the table.

3.3 NLP Module

The NLP module matches properties and columns. It relies on Word2Vec [7], a group of deep learning models which makes it possible to represent words in a vectorial space. We can ask Word2Vec for the cosine similarity of two vectors which represent two words where the cosine similarity is expressed by a float. We assume that the cosine similarity is representative of a semantic proximity of these two words. When we match a column with a property, we tokenize the name of the property and the name of the column. We then compare each token from the column's name with each token from the property's name. The highest cosine similarity is kept as a score for the matching between this column and property.

The NLP module is essential to our process as it is the only one of our modules that can run without data from previous matchings made by the user. This allows

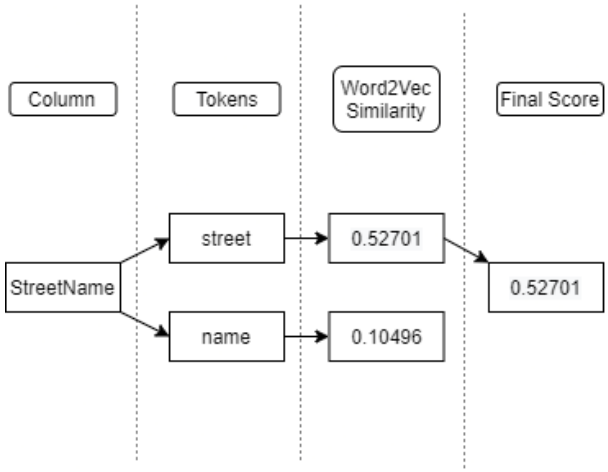

Fig. 2. NLP Module matching column with property "locn:thoroughfare"

us to make predictions on new data. Figure 2 shows how the NLP module scores the match between property 'thoroughfare' and a column named 'StreetName'. In our implementation we chose to use word embeddings trained on Google News articles[3]. When working on a specific domain it may be beneficial to train word embeddings on domain specific documents to ensure the recognition of some words and to prevent the semantics from common parlance from diluting the meaning of words in the domain.

3.4 String module

The string module works by using predicates to establish the profile of the strings in the name and in the content of attributes, each predicate tests a specificity of the string. For example does the string have numbers? special characters? less than X tokens? starts with the prefix X? It is with dozens of such predicates that we can determine a set of labels to represent a string. Each positive predicate generates a label. These labels are known by the module who assigns them weights from the strings that have previously matched each property. Figure 3 shows an example of some predicates that are generated for the shown data.

During its initialization phase, the module retrieves data from previous sessions, and measures the prevalence of each label for each property and compares it to the prevalence of that label in all of the other classes to determine the weight to give it. In general, the more a label is present in the strings of its property and the less it is present in the other properties, the greater its weight will be. A label that would be present in all strings would therefore indicate nothing and will have a weight of zero. From the best results of these modules

[3] https://code.google.com/archive/p/word2vec/

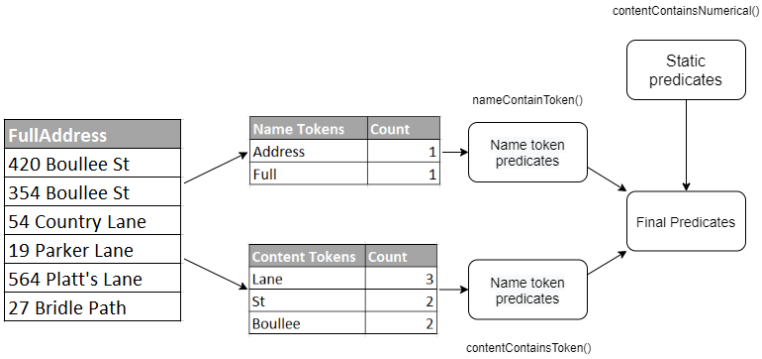

Fig. 3. Tokenization and generation of some predicates from a column and its content This data generates the following numeric predicates among others: nameContainsAddress(), contentContainsNumerical(), contentContainsLane(), contentContainsSt().

on each attribute of the entity, we determine a degree of certainty for the correspondence with this class. The class with the best certainty is kept and the properties which correspond best to its attributes are also preserved. These results are presented to the user who can change the results by choosing the class and properties herself if the results do not suit him.

3.5 Numeric module

The numeric module uses the same mechanics as the string module but analyses the content of columns as integers, floats and doubles instead of simple strings. The predicates built by this module focus on the essential aspects of the numbers present in the column. For instance, the presence of negative values, floating values, and the range of values present in the column are significant information that help us make a clear distinction between concepts like a postal code and a latitude.

3.6 Combining the scores

The final score for the matching between a property and a column is calculated by adding up the scores of the 3 modules. The score of the NLP module is divided by two in this step. This does not impact its performance in cases with no previous data as it is the only active module in these cases. This is done because we think the numeric and string modules make better matchings individually but the NLP module can still help break ties in ambiguous cases.

4 Evaluation

In this section, we present a preliminary evaluation of our solution. We are comparing our results to a well-established semantic data integration tool.

4.1 Data sets and Ontologies

In this experimentation, our goal is to create RDF data from textual documents. We are considering location[4]and geo[5]ontologies. We will use these ontologies to represent addresses using their house street name(locn:thoroughfare), number (locn:poBox), full address (locn:fullAddress), postal code (locn:postalCode) and city name (locn:postalName). The geo ontology will be used to represent geographical points by using their longitude and latitude (geo:long, geo:lat).

We will translate data from four different open data projects from different countries. The "Base Adresse Nationale" (BAN[6]), a national project from the French government, and open data projects from Montreal[7], Austin[8]and London[9]. We cleaned the data before translating it by separating the geographical information to form a "Geographical" table and an "Address" table and ignored data that we would not translate, like ids, keys and metadata from the original databases. The largest case was BAN, where only 7 of the original 20 columns were relevant for our ontologies.

4.2 Comparison setting

Karma [4] is a data integration tool allowing users to generate RDF data from a variety of formats. The tool facilitates the mapping of columns from the source data to properties and classes in ontologies. When mapping columns, Karma recommends a number of mappings based on the content of the column and by learning from matches made by the user in previous sessions. We will be comparing the quality of the top recommendation from our approach with Karma by mapping the same data on the same ontologies using both systems.

4.3 Data and Results

Since our objective is to automate the translation process, we measure the quality of our matching by counting the number of user interactions necessary to create linked data from a table. We increment the counter of user interactions when the proposed matching is incorrect. We count matching a property and a literal as "Assign Property" and matching between a property and a class as "Assign Relationship". Both Karma and our solution start the evaluation with no previous matches to learn from and the matchings expected are the same for both solutions.

Figure 4 details user interactions that were counted when translating data from the different sources into linked data. We reach an average of 0.45 user actions per column using our approach, compared to Karma's 0.68. We notice

[9] https://www.w3.org/ns/locn
[9] https://www.w3.org/2003/01/geo/
[9] https://adresse.data.gouv.fr/donnees-nationales
[9] https://donnees.montreal.ca/ville-de-montreal
[9] https://data.austintexas.gov/Locations-and-Maps/
[9] https://opendata.london.ca/

Source	Number of Rows	Table Name	Columns	User Actions					
				Karma			Our Process		
				Assign Property	Assign Relationship	Total	Assign Property	Assign Relationship	Total
BAN	25 000 000	Address	5	4	1	5	2	1	3
		Geographical	2	2	0	2	1	0	1
Austin Open Data	425 780	Address	3	0	1	1	2	0	2
		Geographical	2	2	0	2	0	0	0
London Open Data	135 518	Address	4	2	0	2	2	0	2
Montreal Open Data	343 278	Address	4	1	0	1	1	1	2
		Geographical	2	2	0	2	0	0	0
Average User Interaction per Column =				0.681818182			0.454545455		

Fig. 4. Results for the initial matching

that out of the 7 properties that we were matching with the data, 3 were found solely based on NLP in the first dataset.

5 Related Work

Most solutions for ontology matching and mapping are either mapping languages or transformation software. In both cases, they are oriented towards expert users like ontology experts. Karma stood out as it was trying to be accessible to anyone with an understanding of their data and their ontology with a semi-automated approach. We based the core of our approach on predicates inspired from their approach and expanded it by introducing NLP, the addition of NLP allows us to suggest mappings on properties and classes that have not been matched before. This was not possible with Karma as it needed data from previous matchings made by the user for a property before suggesting it in new matchings. The other addition we made to karma's process was to split predicates on content between string based predicates and numerical predicates.

6 Conclusion and future work

We have proposed an approach to semi-automatically match non-semantic data with ontologies by using machine learning and NLP techniques to create knowledge graphs. We have demonstrated that combining different approaches, in particular NLP and string matching, improve the quality of our matchings. The quality of our matchings depends directly on the quality of the data we are matching and the quality of the data from the previous sessions. Data attributes using real life vocabulary can be directly used to suggest matchings with our NLP approaches whereas truncated words, acronyms are unusable. The more we use each ontology, the better the quality of our matchings.

Even though our process shows good results, it could be improved by extending its NLP approaches. In our evaluation several sources had their schemas in different languages which complicated the task of our NLP module as Word2vec must have seen the exact word being matched to give an answer. If we translated the names in the data's schema in a separate phase or if the NLP module was capable of doing it automatically, its results should improve significantly. The combination of the different modules itself could be improved by adjusting the

weight of the different scores based on how accurate their predictions were for a given property in previous sessions. The model that we train will be made reusable and able to learn during sessions in future works, this will improve the quality of our matchings during the sessions themselves. In this paper our focus was on the quality of the matchings, this focus however came at a cost when it comes to memory consumption and processing power, work will be needed to optimize our approaches to handle larger amounts of data. The full integration of the graphical user interface and the extension of its functionalities will be essential. We are working towards the development of a full application.

References

1. Dimou, Anastasia & Vander Sande, Miel & Colpaert, Pieter & Verborgh, Ruben & Mannens, Erik & Van de Walle, Rik. (2014). RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. CEUR Workshop Proceedings. 1184.
2. Lefrançois, Maxime & Zimmermann, Antoine & Bakerally, Noorani. (2017). Flexible RDF Generation from RDF and Heterogeneous Data Sources with SPARQL-Generate. 131-135. 10.1007/978-3-319-58694-6_16.
3. S. Das, S. Sundara, and R. Cyganiak. R2RML: RDB to RDF Mapping Language. W3C Recommendation. https://www.w3.org/TR/r2rml/, September 2012.
4. Knoblock C.A. et al. (2012) Semi-automatically Mapping Structured Sources into the Semantic Web. In: Simperl E., Cimiano P., Polleres A., Corcho O., Presutti V. (eds) The Semantic Web: Research and Applications. ESWC 2012. Lecture Notes in Computer Science, vol 7295. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-30284-8_32
5. Codescu, Mihai Horsinka, Gregor Kutz, Oliver Mossakowski, Till Rau, Rafaela. (2014). OSMonto -An Ontology of OpenStreetMap Tags.
6. F. Cerbah. Learning highly structured semantic repositories from relational databases: The RDBToOnto tool. In Proceedings of the 5th European Semantic Web Conference (ESWC 2008), pages 777–781, Athens, GA, USA, 2008. ISBN 3-540- 68233-3 978-3-540-68233-2.
7. Mikolov, Tomas Corrado, G.s Chen, Kai Dean, Jeffrey. (2013). Efficient Estimation of Word Representations in Vector Space. 1-12.