Browsing Linked Open Data with Auto Complete

Heiko Paulheim

Technische Universität Darmstadt
paulheim@ke.tu-darmstadt.de

Abstract. Information in Linked Open Data is incomplete by nature and design. Nevertheless, the more complete information delivered on a subject is, the higher is its value to an end user. With our submission to the Semantic Web Challenge[1], we show how information, in particular types, can be completed automatically, based on heuristic rule learning. We introduce an approach which employs *lazy learning* instead of learning a global model, making the approach well scalable to large data, while at the same time providing results at an accuracy of 85.6%. The auto complete function is included in a modular semantic web browser.

Keywords: Linked Open Data, Automatic Completion, Association Rule Mining, Ontology Learning, Ontology Alignment, Lazy Learning

1 Introduction

Information in Linked Open Data is incomplete by nature and design. Complete information about a subject can rarely be provided with a reasonable amount of efforts. Thus, the open world assumption underlying RDF and Linked Open Data always assumes incomplete information.

While creating and providing information can be expensive, more complete information may have higher value for end users. To resolve this dilemma, we propose an automatic approach for completing information, in particular type information.

In recent works, machine learning approaches have been proposed to complete missing information in Linked Open Data. However, they can either focus on only one particular problem (e.g., learning the missing value for one property [**?**]), with the need to user invention by manual selection of relevant information, or use only a small sample of the whole data [**?**]. So far, approaches that are able to cope with arbitrary completion problems in a completely unsupervised manner can rarely handle the large amount of data and the size of models to learn.

In our submission to the Semantic Web Challenge, we show a Linked Open Data browser that automatically completes Linked Open Data on the fly, when displaying a resource. The underlying algorithm automatically completes type information by using association rule mining. While classical machine learning algorithms learn a complete,

[1] Live demo: http://kebap.ke.informatik.tu-darmstadt.de:8080/LODATC/

global model, lazy learning algorithms learn partial models only for instances of interest [**?**]. For the prototype, we have thus implemented a lazy variant of the apriori algorithm for association rule mining [**?**], which is fast enough to be embedded in a Linked Open Data browser.

2 Approach

In this section, we use the example of Pete Stewart, an American musician. In the corresponding DBpedia entry, there are, among others, the following triples[2]:

```
dbpedia:Pete_Stewart  a yago:Singer110599806        (1)
dbpedia:Pete_Stewart a yago:AmericanMusicians       (2)
```

On the other hand, an example for a missing triple is

```
dbpedia:Pete_Stewart a yago:AmericanSingers         (3)
```

If there was an ontology-level statement stating

$$\texttt{yago:Singer110599806} \sqcap \texttt{yago:AmericanMusicians} \sqsubseteq \texttt{yago:AmericanSingers}, \quad (4)$$

the missing triple could be easy completed. However, most of the ontologies used in Linked Open Data do not provide that rich level of axiomatization. Thus, that type cannot be obtained by simple computation of the transitive closure of all the instance's types.

To overcome this shortcoming, ontology learning approaches have been proposed. We follow the method discussed in [**?**], proposing the use of association rule mining [**?**]. The apriori algorithm takes a table of instances as input, with the types listed for each instances (called item sets). It then tries to find patterns of co-occuring types. In the example above, if there are enough examples in the knowledge base that have the types `yago:Singer110599806`, `yago:AmericanMusicians`, and `yago:AmericanSingers`, the pattern can be found.

However, learning a complete set of association rules on the whole of Linked Open Data and applying that rule set would lead to immense computational efforts. In order to create a more scalable approach, we have implemented the apriori algorithm in a lazy variant [**?**], which creates only those rules that are relevant for a single instance. For example, the rule depicted in equation **??** could be learned for the instance `dbpedia:Pete_Stewart` by starting a search from the types given for that instance.

Once the rules are learned, we compute the set of possible types from the rules found for the instance. The apriori algorithm provides a confidence value for each rule, which we set as the confidence value for the type statement derived from that rule. In case more than one rule states that a type should be present, the confidence values are combined following the approach discussed in [**?**]:

$$confidence(t) := 1 - \prod_{\text{all rules r with t in head}} 1 - confidence(r) \quad (5)$$

[2] see `http://dbpedia.org/resource/Pete_Stewart`

The rationale behind this formula is that if there may many pieces of weak evidence (i.e., rules with low confidences) for a type, this may be a good indicator for a type as well as one rule with high confidence.

Without further corrections, this approach would lead to many false positives. For example, many people in Wikipedia are athletes, which leads to rules such as $dbpedia : Person \sqsubseteq dbpedia : Athlete$. To overcome such problems, we regard all types that are explicitly given as true. Each type found which is disjoint with one of the explicit types can thus not be true and is discarded. Since there rarely are disjointness axioms in ontologies used in Linked Open Data, we regard two types to be implicitly disjoint if there are no common instances in the whole data set. For the example above, there are no instances in DBpedia that are both of type `dbpedia:Athlete` and `yago:AmericanMusicians`. Thus, the type would be discarded.

The types that are found for a resource may not not only be disjoint to the resource's explicit types, but also to each other. In that case, the type with the lower confidence is discarded. In the example above, both `yago:AmericanMaleSinger` and `yago:AmericanFemaleSinger` are found, which are disjoint. Since the first has the higher confidence, the latter is discarded.

It is noteworthy that the approach does not only generate types within the ontologies that are used for the specific instance, but also for other ontologies. In the example above, our approach also generates the type `umbel:MusicalPerformer`, although no mappings to the UMBEL ontology are given for the respective resource. Thus, it does not only complete type information within one data set, but also adds mappings to other data sets.

3 Prototype

We have integrated our approach for automatic type completion as an extension into the modular Linked Open Data browser *MoB4LOD*[3]. The browser has been configured to show three panels: the direct types given for an instance, the types automatically completed, including scores, and the remaining triples about the instance. Fig. **??** shows a screenshot of the user interface.

The user interface provides an input field where the user can enter a URI or a keyword, which is resolved into a DBpedia URI using the DBpedia Lookup Service[4]. Furthermore, the user can navigate between instances by clicking on links in the statements table.

4 Evaluation

We have evaluated the quality of results of our approach using 50 randomly sampled instances from DBpedia. For the parameters of the apriori algorithm, we have set $support_{min}$ to 0.1 (where the support of a rule is computed relative to the smallest type in the set) and $confidence_{min}$ to 0.05. Furthermore, we have discarded all types

[3] `http://www.ke.tu-darmstadt.de/resources/mob4lod/`

[4] `http://wiki.dbpedia.org/lookup/`

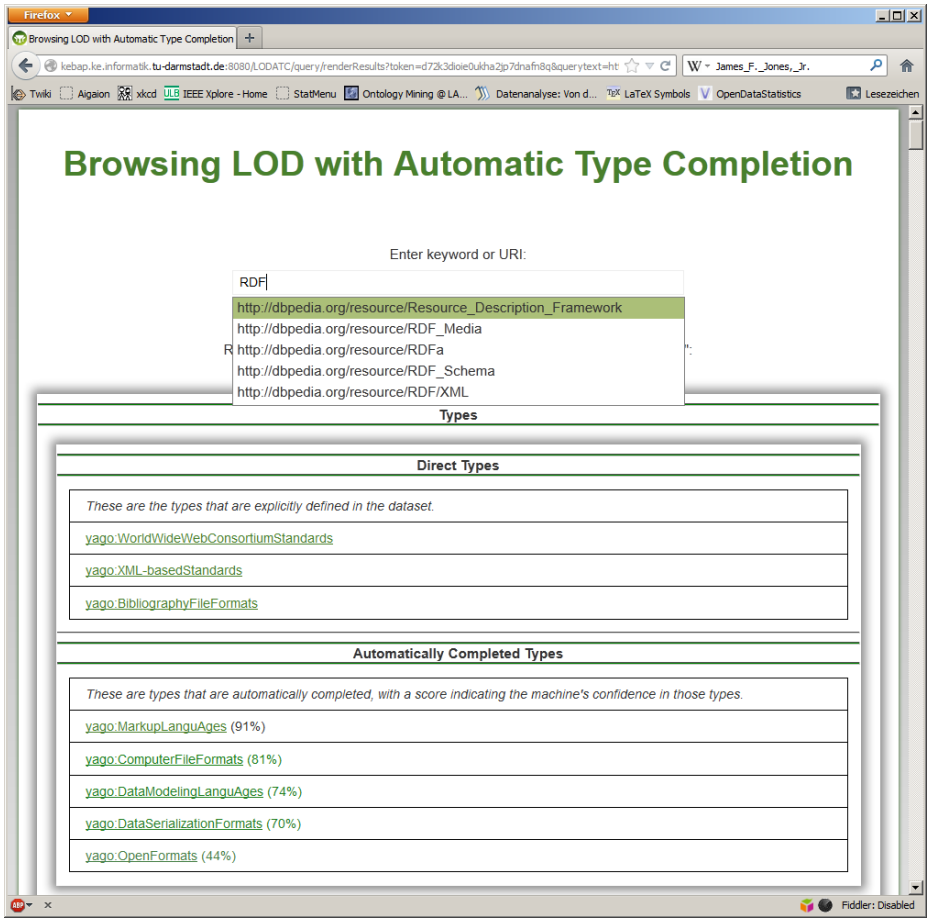

Fig. 1. Screenshot of the browser including autocompleted types

below an overall confidence of 0.15. The rationale of having two confidence values is that according to (**??**), some weak pieces of evidence (i.e., rules) can multiply to a larger overall confidence value, so we did not want to discard them too early. We varied the maximum size of the item sets explored by the apriori algorithm.

As a baseline, we computed the transitive closure of all types of an instance. A general observation is that except for rare exceptions, the types generated the baseline and by our method are disjoint. Using the transitive closure leads to very general types such as `Object`, `Entity`, or `Thing`, which are of limited interest for the user. In contrast, our approach produces more concise types, as well as types from other ontologies, as discussed above, which are rarely to be found by simply generating the transitive closure due to the low spreading of schema-level mappings in Linked Open Data.

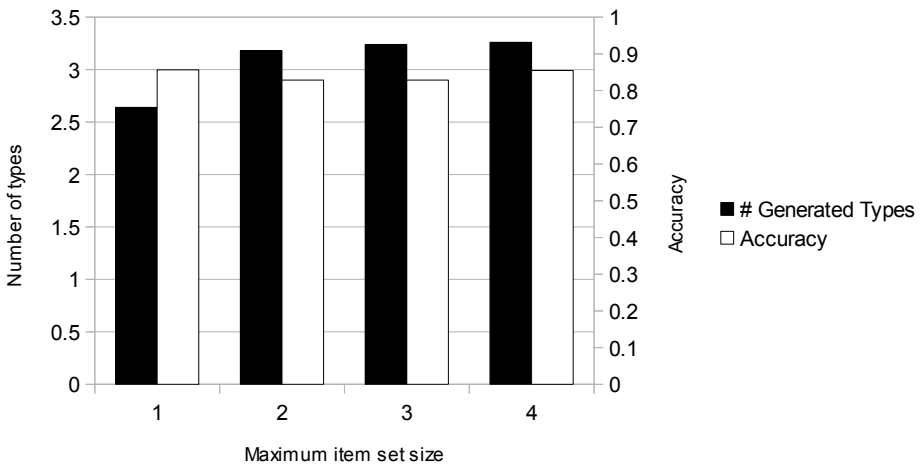

Fig. 2. Quality of the generated types for different maximum item set sizes

Figure **??** shows the average precision and number of types generated by our approach, depending on the maximum subset size. It can be observed that up to 3.26 additional types are generated on average, with an accuracy between 82.8% and 85.6%.

In contrast, 4.52 additional types are created by the baseline, which are always correct. Since the types generated by both approaches are disjoint except for very rare exceptions, a total of 7.78 additional types can be created[5].

Figure **??** depicts an analysis of the runtime behavior of our approach. It can be observed that the runtime grows about linearly with the size of item sets, so does the number of item sets examined, each examination requiring an additional SPARQL query (the decrease from a maximum itemset of three to four is not statistically significant). When the number of item sets is kept low (i.e., at one and two), there are reasonable run times for an interactive application. While the average run-times for one and two are at around 1700 and 2800 milliseconds, the median is much lower (611 and 606 milliseconds, respectively), while a few resources take much longer to process. Overall, a good trade-off between result quality, number of types generated, and run-time efficiency is reached when setting the maximum item set size to two.

5 Conclusion and Outlook

With this contribution to the Semantic Web Challenge, we have introduced a prototype for exploring Linked Open Data with automatic, on-the-fly completion of missing statements. The results show that missing statements can be completed with a high quality and within reasonable amounts of time for an interactive application.

So far, our approach is restricted to inducing type statements, based on other type statements. For the future, we aim at using other features for learning (such as the

[5] However, we have not included the results from the baseline in the web demonstrator, since they are mostly uninteresting, as discussed above.

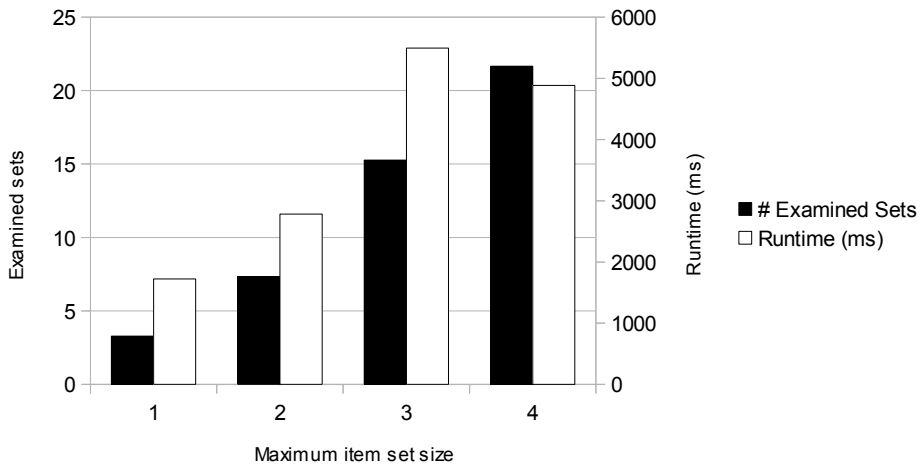

Fig. 3. Runtime behavior of the approach for different maximum item set sizes

types of related instances), as discussed in [**?**], as well as inferring missing statements other than types. Furthermore, providing justifications of the inferred statements (by explicating the inferred rules) to the user would enhance the user experience of the browser.

The auto complete mechanism could also be of value when being plugged into an editor for Linked Open Data, giving the author of resources hints for statements that are potentially missing.

Acknowledgements

The author would like to thank Chinara Mammadova, Dominik Wienand, Jan Stengler, Peter Klöckner and Melanie Weiland for their support in implementing the browser framework.

Appendix

This appendix describes how our submission meets the challenge criteria.

Minimal Requirements

End-user Application The application is a Linked Open Data browser targeted at users of the semantic web. It provides additional value for browsing the semantic web.

Information Sources The approach is generic and works with any kind of Semantic Web data, following elementary standards such as RDF.

Meaning of Data By augmenting data with additional type information, the application provides more insights into the semantics of the viewed resources.

Additional Desirable Features

Web Interface The application provides a functional web interface, which we hope is attractive.

Scalability By implementing a lazy learning algorithm which only processes required data on the fly, we can cope even with large datasets such as DBpedia in a reasonable amount of time. A scalability evaluation is included in the paper.

Evaluation We have evaluated our approach with respect to result quality and runtime.

Accuracy and Ranking of Results The application provides its information ranked by confidence, and makes the confidence values explicit.