An On-Line Learning to Query System

Jedrzej Potoniec

Faculty of Computing, Poznan University of Technology
ul. Piotrowo 3, 60-965 Poznan, Poland
Jedrzej.Potoniec@cs.put.poznan.pl

Abstract. We present an on-line system which learns a SPARQL query from a set of wanted and a set of unwanted results of the query. The sets are extended during a dialog with the user guided by *recall* and F_1 *measure*. The system leverages SPARQL 1.1 and does not depend on any particular RDF graph.

1 Introduction

A common problem with querying a Linked Data dataset is that the user must have prior knowledge about the vocabulary used by the dataset and know a querying language, e.g. SPARQL [3]. A typical approach to remedy the problem is to use some tool helping the user to formulate a SPARQL query, using e.g. faceted browsing [2], natural-language interfaces [4], visual interfaces [7] or recommendations [1]. In all these system the user specifies the query by various means. We propose a different approach, where the user only specifies what should and what should not be in the results of the query, and the system takes care of formulating the query. The user does not need to know SPARQL, it is enough for her to be able to distinguish wanted and unwanted results. If the user is already familiar with the data (e.g. knows its representation in some other format), it is quite an easy task for her. The system operates by conducting a dialog with the user. In each part of the dialog, the user is asked about a small set of URIs and for every URI she must decide if it should or should not be present in the results of the final query. A similar approach was already presented in [5], where the system used mostly computational power of the computer system of the user. We leverage new features of SPARQL 1.1 and move most of the computation to a SPARQL endpoint, especially the process of query refinement guided by *recall* and deciding on termination depending on F_1 *measure*.

Thought this work, we use the following prefixes: `dbr:` for `http://dbpedia.org/resource/`, `dbo:` for `http://dbpedia.org/ontology/`, `dbp:` for `http://dbpedia.org/property/`, `dct:` for `http://purl.org/dc/terms/`, `xsd:` for `http://www.w3.org/2001/XMLSchema#`.

2 System Description

A screenshot of the on-line system is presented in Figure 1. The source code of the system is available in a *Git*[1] repository available at `https://bitbucket.org/jpotoniec/kretr/`. An instance of the system is available at `https://semantic.cs.put.poznan.pl/ltq/`. It uses a SPARQL endpoint set up on *Blazegraph*[2] *2.1.1* and loaded with *DBpedia 2015-04*. Note that the system itself does not depend neither on *Blazegraph* nor *DBpedia*, as it can use any SPARQL 1.1 endpoint.

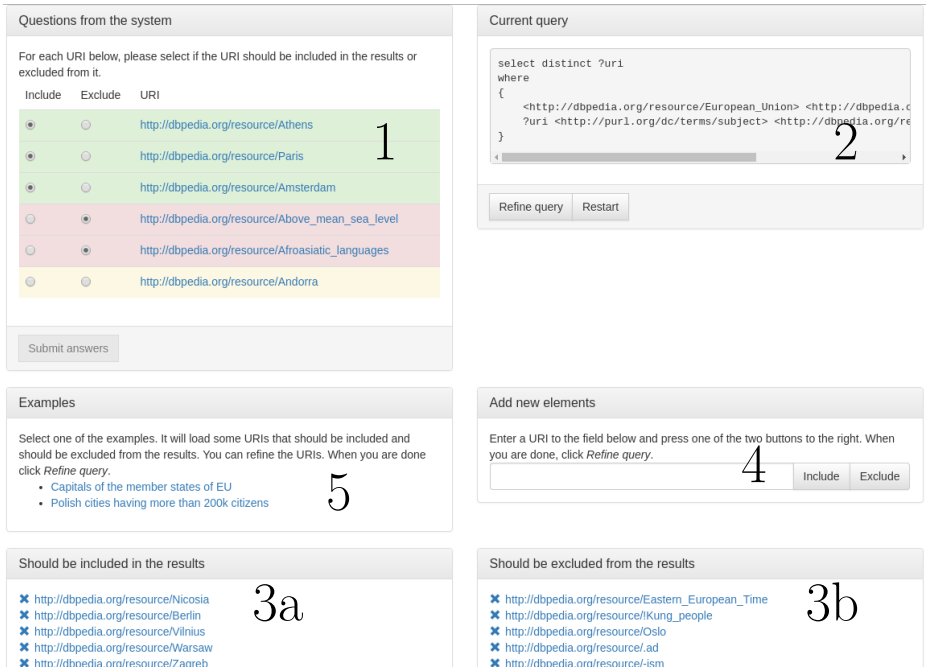

Fig. 1. A screenshot of the system while the user is supposed to assign new examples to correct sets. The view is divided into six areas: new examples, which are to be assigned (1), the current query (2), the URIs already assigned by the user: wanted (3a) and unwanted (3b), a form to add a new URI by hand (4) and demo scenarios (5).

The aim of the system is to build a SPARQL SELECT query with a single variable in the head. The variable has a modifier DISTINCT. The query contains only a WHERE clause (i.e. there is no GROUP BY, ORDER BY etc.), and in the clause there is only a basic graph pattern (BGP, i.e. triple patterns and filter expressions). The undirected graph corresponding to the BGP is a connected

[1] `https://git-scm.com/`
[2] `https://www.blazegraph.com/`

graph and the filter expressions are of a form *variable* `>=/<=` *literal*. An example of such a query is

```
SELECT DISTINCT ?uri
WHERE {
    ?uri dct:subject dbr:Category:City_counties_of_Poland .
    ?uri dbo:populationTotal ?anon1.
    FILTER(?anon1 >= "205934"^^xsd:nonNegativeInteger). }
```

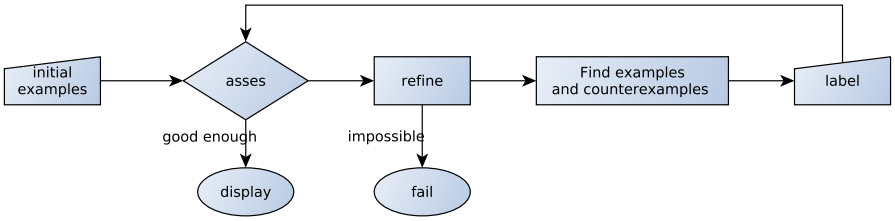

Fig. 2. A typical workflow with the system. First the user specifies a small set of examples, then the system works in a loop: asses a hypothesis, refines it, finds new examples and asks the user to decide about them.

A typical workflow with the system is presented in Figure 2. First, the user specifies a small set of URIs that should be present in the results of the final query, and a small set of URIs that should not be present in the results. Then, the system refines the query by adding to it a new triple pattern (or a triple pattern and a filter expression) while maintaining recall of at least 0.99, i.e. covering at least 99% of the positive examples. In a typical case of multiple possible refinements, they are sorted by F_1 *measure* and *precision* and the one with the highest values is chosen. Next, the system generates a few new positive and negative examples. The positive examples are simply selected from the results of the refined query, while the negative examples are computed by subtracting from the results of the original query the results of the refined query. In each case we require the found examples to be new, i.e. they must not be already labeled by the user. If finding the new examples is impossible, the refinement is retracted and the next in order is tried. The user is then asked to decide about each example if it should or not be present in the results of the final query (i.e. the user extends the sets defined in the beginning). After the user decides, the system checks if the decisions agrees with the query. If they do not, the cycle repeats: the system refines the query, asks the user and assess the query. Otherwise, the system assumes that the correct query was found. The query is displayed to the user along with the results of the query. If the user decides that the results are not satisfactory, she must add at least one new URI to one of the sets. More details about the algorithm, along with the templates of queries used, is presented in [6].

3 Proposed Demo

During the demo, we will present to the participants how to use the system, what types of queries are possible to learn and what are the limitations. The system has embedded two demo scenarios: the first one to find a query to select all capitals of the member states of European Union and the second one to find a query to select all Polish cities having more than 200′000 citizens. For the presentation, we will use the on-line instance available at `https://semantic.cs.put.poznan.pl/ltq/`.

4 Conclusion

In this paper we presented a system which is able to learn a SPARQL query from two sets of URIs obtained from the user in a dialog. The system leverages new features of SPARQL 1.1 and does not depend on any particular RDF graph. It also does not require any precomputation before it is ready to use. The source code is publicly available.

Acknowledgement. Jedrzej Potoniec acknowledges the support from the Polish National Science Center (Grant No 2013/11/N/ST6/03065).

References

1. Campinas, S.: Live SPARQL auto-completion. In: Horridge, M., Rospocher, M., van Ossenbruggen, J. (eds.) Proc. of the ISWC 2014 Posters & Demonstrations Track. CEUR Workshop Proceedings, vol. 1272, pp. 477–480 (2014)
2. Ferré, S.: SPARKLIS: a SPARQL endpoint explorer for expressive question answering. In: Horridge, M., Rospocher, M., van Ossenbruggen, J. (eds.) Proc. of the ISWC 2014 Posters & Demonstrations Track. CEUR Workshop Proc., vol. 1272, pp. 45–48 (2014)
3. Harris, S., Seaborne, A.: SPARQL 1.1 query language. W3C recommendation, W3C (Mar 2013), http://www.w3.org/TR/2013/REC-sparql11-query-20130321/
4. Höffner, K., Walter, S., Marx, E., Usbeck, R., Lehmann, J., Ngomo, A.C.N.: Survey on challenges of question answering in the semantic web. Semantic Web Journal (accepted for publication) (2016)
5. Lehmann, J., Bühmann, L.: AutoSPARQL: Let users query your knowledge base. In: Antoniou, G., Grobelnik, M., et al. (eds.) The Semantic Web: Research and Applications. LNCS, vol. 6643, pp. 63–79. Springer (2011)
6. Potoniec, J.: Learning to Query: from Concepts in Mind to SPARQL Queries. Tech. Rep. RA-9/16, Institute of Computing Science, Faculty of Computing, Poznan University of Technology (aug 2016), `http://goo.gl/B7J008`
7. e Zainab, S.S., Saleem, M., Mehmood, Q., et al.: Fedviz: A visual interface for SPARQL queries formulation and execution. In: Ivanova, V., Lambrix, P., et al. (eds.) Proc. of the International Workshop on Visualizations and User Interfaces for Ontologies and Linked Data. CEUR Workshop Proc., vol. 1456, p. 49 (2015)