DBpedia Navigator

Jens Lehmann and Sebastian Knappe

Universität Leipzig, Department of Computer Science, Johannisgasse 26,
D-04103 Leipzig, Germany,
lehmann@informatik.uni-leipzig.de, sebastian_knappe@gmx.de

Abstract A vision of the Semantic Web is to bring the benefits of semantic technologies to a wide audience. Large knowledge bases, such as DBpedia, YAGO, and others have emerged and are freely available as Linked Data and SPARQL endpoints. In this article, we introduce the application *DBpedia Navigator*, which allows users to navigate, search, and browse within DBpedia and the data sets interlinked with DBpedia. In particular, we present a novel feature called *navigation suggestions*. Based on the last instances viewed in DBpedia Navigator, the user gets suggestions about related instances he may be interested in. Those suggestions make full use of the semantics of the underlying knowledge base and are provided by a supervised Machine Learning algorithm for OWL, which we made available as open source in the DL-Learner project.

1 Introduction

The vision of the Semantic Web is to make use of semantic representations on the largest possible scale - the Web. We currently experience that Semantic Web technologies are gaining momentum and large knowledge bases such as DBpedia[1], YAGO[6], and others are freely available. These knowledge bases are based on semantic knowledge representation standards like RDF and OWL. They describe millions of objects, and contain hundred thousands of properties as well as classes.

Due to their sheer size, users of large knowledge bases, however, are facing the problem, that they can hardly know which identifiers are used and available. In most cases, end users will not be able to express their queries in a structured form at all, but they often have a very precise imagination what kind of results they would like to retrieve. A historian, for example, searching for ancient Greek law philosophers influenced by Plato in DBpedia can easily name some examples and if presented a selection of prospective results he will be able to quickly identify false results. However, he might not be able to efficiently construct a formal query adhering to the large DBpedia knowledge base a priori.

The DBpedia Navigator is an application, which tries to overcome this problem and facilitate browsing and querying large knowledge bases. It was written as an interface to DBpedia and the data sets interlinked with it, but can in principle be configured to be used in conjunction with arbitrary large knowledge bases available as SPARQL endpoints. It tackles the issue mentioned above by

using Machine Learning techniques, which offer the user *navigation suggestions*. Furthermore, it provides many other means to browse through the DBpedia knowledge base taking the underlying semantics into account.

The paper is structured as follows: There are two main sections. Section 2 describes the DBpedia Navigator user interface and the structure as well as features of the application. Section 3 describes the working of the navigation suggestion component, which is based on the DL-Learner[1] Machine Learning framework. Finally, in Section 4, we give concluding remarks and pointers to future work.

2 The DBpedia Navigator User Interface

DBpedia Navigator is available at `http://navigator.dbpedia.org`. In brief, it is a user interface for browsing, searching, and navigating within DBpedia. While the application will later be suitable for end users, it is currently in Alpha status, i.e. it is availabe for testing purposes, but is not yet sufficiently stable and complete to be used productively. In the following, we present its user interface structure.

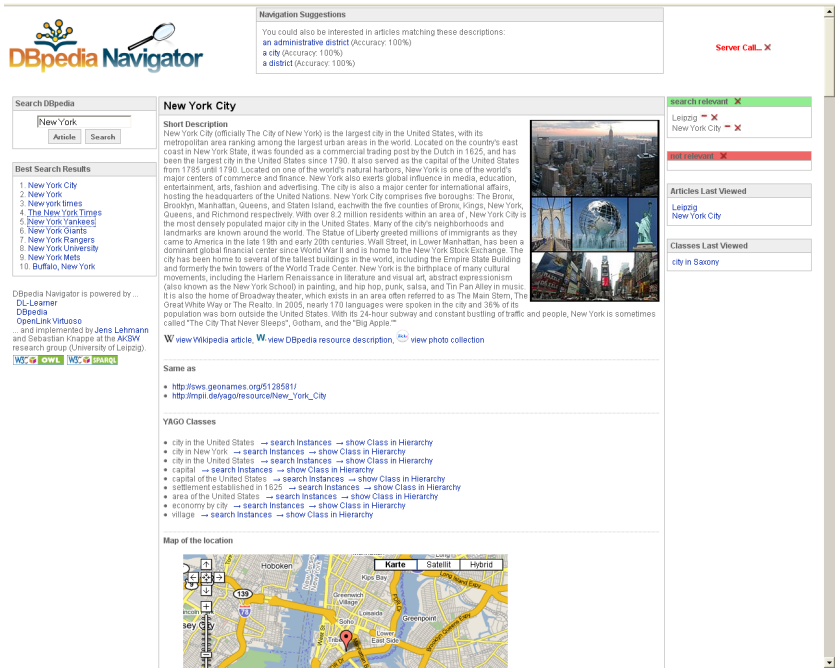


Figure 1. The DBpedia Navigator GUI

[1] `http://dl-learner.org`

2.1 Left Sidebar

You can search for an article by entering its name into the search field, which is located on the left side (see Figure 1). We use the phrase article, because the displayed information is to a large extend derived from Wikipedia articles. Contrary to a Wikipedia display of an article, we view an article as a collection of content connected to an object. There are two ways to search an article: Hitting the `Article` button performs a direct article search, i.e. looks for a perfect match. Using the `Search` button performs a search for articles which have labels similar to the search string. They are presented in a search result view. DBpedia Navigator keeps the results of your latest search on the left sidebar even after you selected a search result.

2.2 Center Section

All views are shown in the center box. There are three kinds of views: article view, search result view, and a class view.

Article View For the article view the label of the current object is shown in the upper left corner of the box. The article itself consists of several parts, which are separated by a dotted horizontal line. The first part shows a short description of the object with an associated picture. Links to the corresponding Wikipedia article, the DBpedia resource description and a Flickr photo collection are given, which are opened in small windows, so that you can look at the article and the opened link at the same time.

The second part consists of links to interesting interlinked objects. Below those are the YAGO classes of the object. Using the displayed links, you can search for instances of these classes, which takes you to a search result view of the instances, or alternatively you can view information about the class itself. The fourth part is a content-specific section depending on the kind of viewed object. If the object is a location, a Google Map is shown, if it is a person, some characteristics of that person are displayed etc.

The next part of the article view is a collection of interesting information, which was not yet consumed by any of the parts above (interlinked data, class hierarchy, short abstract etc.) and is not ignored by a configurable object filter (e.g. we filtered out SKOS properties since we use YAGO). They are displayed in a style similar to typical linked data browser. If there are more than a configurable amount (default: 3) of objects associated with a property, only the first are shown. The remaining objects can be made visible via a show button. This feature makes the article display more compact. We are aware that large fractions of the displayed information in this part is not suitable for direct consumption by end users, because DBpedia - while being an immensely useful resource - does (can) not reach the quality of a maintained ontology (yet).

Shown articles are automatically added to the list of search relevant articles in the upper box on the right sidebar. These instances are used to generate navigation suggestions (see Section 3).

Figure 2. Display of search result along with a tag cloud to restrict results.

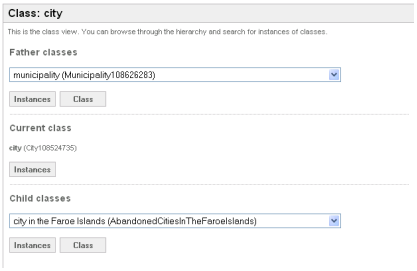


Figure 3. The class view contains parents, children, and instances of a class.

Search Result View The search result view is shown whenever you search for articles, by label, by class or by class description. If you search for articles using a label a tag cloud is displayed, showing the classes, the found articles belong to, with the option to filter the articles by these classes. What follows regardless of what kind of search you performed is a list of links to articles on several pages, with the option to show a variable number of results. The results are ordered according to a pagerank, every article has. Therefore you always see the most relevant articles, matching a certain criteria.

Class View The class view is a view of a class in the context of the class hierarchy, that means you see superclasses and subclasses of the searched or selected class. You can select classes in this view and query their instances. This allows you to browse not only via articles but also via the class hierarchy, enabling more complex and sophisticated use of the application.

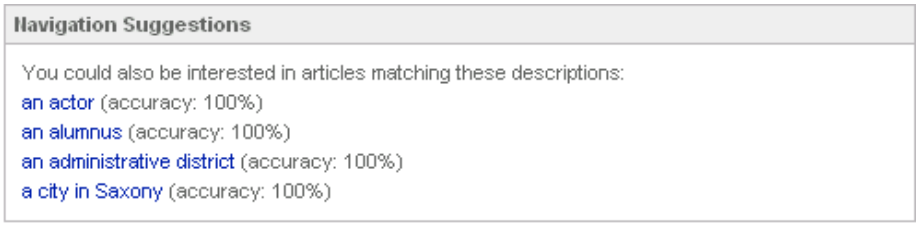
2.3 Right Sidebar

The right side bar consists of up to four elements. On the upper right side there are two boxes with lists of search relevant and not relevant articles. You can move articles from “relevant” to “not relevant” or the other way around by simply clicking on the plus or minus behind the articles. The cross removes articles from these boxes. The lists are important when generating navigation suggestions, since they correspond to positive and negative examples of the learning problem explained later on. In short, those navigation links are suggested, which cover all relevant and none of the irrelevant articles. Below the relevance boxes the last shown articles and classes are displayed. To disburden the SPARQL endpoint, they are kept in a cache.

2.4 Navigation Suggestions

The navigation suggestions are shown above the articles. They are links, that call a function to show instances of the class, these links are representing. The

classdefinition is not shown in description logic but in a human readable way. For every suggestion the accuracy is noted, indicating how good a class covers the given examples. On the right side is an indication whenever the server is called and you can cancel a request if it takes too long.



2.5 REST Interface

The application has a REST Interface, which allows to map the current state of the application to an URL. The URL is shown below the article and is updated everytime you perform an action. The interface has the following syntax:

```
URL        ::= 'http://' host '/' function '/'
label ['?' parameters]
host       ::= (the host, the navigator is installed on)
function   ::= 'showArticle' | 'search' | 'showClass' |
'searchInstances' | 'searchConceptInstances'
label      ::= (some label of the article, the searchphrase or the class)
parameters ::= parameter ['&' parameter]
parameter  ::= ('positives=' article+) | ('negatives=' article+) |
('concept=' concept)
article    ::= '[' uri ']'
uri        ::= (the URI of the article)
concept    ::= (class description)
```

3 Generating Navigation Suggestions

3.1 Overview of the underlying Machine Learning Algorithm

In this section we will briefly describe the learning problem in OWL. We assume familiarity with OWL or Description Logics (DLs) [2]. The process of learning in logics, i.e. finding logical explanations for given data, is also called *inductive reasoning*. The background knowledge is a knowledge base $\mathcal{K}$. The goal is to find a definition for a class we want to call `Target`. Hence the examples are of the form `Target`(a) where a is an individual. We are then looking for a class definition of the form `Target` $\equiv C$ such that we can extend our knowledge base

by this definition. Let $K' = \mathcal{K} \cup \{\texttt{Target} \equiv C\}$ be this extended knowledge base. Then we want that the positive examples follow from it, i.e. $\mathcal{K}' \models E^+$, and the negative examples should not to follow, i.e. $\mathcal{K}' \not\models E^-$.

In inductive learning, a generate and test strategy is common. Several class descriptions[2] are tested during a learning process, each of which is evaluated using an OWL reasoner. Smart algorithms will take the results of those evaluations into account to suggest further promising descriptions. This way, learning can be seen as a search process in the space of descriptions. A natural idea is to impose an ordering on this search space (the set of all class descriptions) and use operators to traverse it. This strategy is well-known in ILP, where refinement operators are widely used to find hypotheses. Formally, if S be the set of OWL class descriptions, we can consider the quasi-ordered space $(S, \sqsubseteq)$ ($\sqsubseteq$ denotes subsumption). A *downward (upward) refinement operator* ρ is a mapping from S to 2^S, such that for any $C \in S$ we have that $C' \in \rho(C)$ implies $C' \sqsubseteq C$ ($C \sqsubseteq C'$). C' is called a *specialisation* (*generalisation*) of C.

We used this idea to build a top-down refinement operator based algorithm. This means that the first description, which will be tested is the most general description (`owl:Thing`, denoted by $\top$ in DL syntax), which is then mapped to a set of more special descriptions by means of a downward refinement operator. Naturally, the refinement operator can be applied to the obtained descriptions again, thereby spanning up a search tree. The search tree can be pruned when we reach an incomplete description, i.e. a description which does not cover all the positive examples. This can be done, because the downward refinement operator guarantees that all refinements of this description will also not cover all positive examples and therefore cannot be solutions of the learning problem. One example for a path in a search tree spanned up by a downward refinement operator is:

$$\top \rightsquigarrow \texttt{Person} \rightsquigarrow \texttt{Person} \sqcap \texttt{participatesIn.Event}$$
$$\rightsquigarrow \texttt{Person} \sqcap \texttt{participatesIn.Conference}$$

The heart of such a learning strategy is to define a suitable refinement operator and an appropriate search heuristics for deciding which nodes in the search tree should be expanded. The refinement operator in the considered algorithm can be found in [5] and is build on solid theoretical foundations [4]. It has been shown to be the best achievable operator with respect to a set of properties (not further described here), which are used to assess the quality of refinement operators.

3.2 Scaling the Algorithm to Very Large Knowledge Bases

Since the algorithm depends on reasoning and OWL reasoners usually are not able to handle very large knowledge bases, such as DBpedia, we first perform a knowledge fragment selection. The extraction works by executing SPARQL queries, which obtain knowledge related to the example instances. The fragment

[2] see `http://www.w3.org/TR/owl-ref/#ClassDescription` for a definition of OWL class description

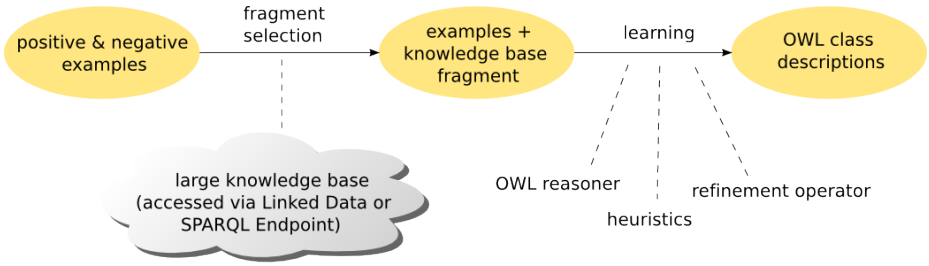


Figure 4. Process Illustration: In a first step, a fragment is selected based on instances from a knowledge source and in a second step the learning process is started on this fragment and the given examples.

is much smaller than the original knowledge base and makes reasoning feasible. The extraction procedure itself is described at [3] and will not be presented in detail here. In general, the extraction procedure starts with the example instances (relevant and irrelevant articles in our case) and finds related instances up to some recursion depth. It then extracts important fractions of schema information. The extraction also handles OWL DL conformance issues e.g. proper typing of resources to ensure that reasoners can properly process the fragment.

3.3 Adapting the Machine Learning Algorithm

The above described learning algorithm [5] was adapted for generating navigation suggestions in DBpedia. First the positive and negative examples are combined to sets of examples, because generating descriptions with examples from very different areas results in very general descriptions, that are of no use as navigation suggestions. For instance, using a location and a person usually do not have much in common. Therefore, the classes and the super classes of example instances are examined, such that if two examples share a class or super class, they are in the same example set. Every example set forms one learning problem, that is solved seperatly. Finally, the resulting suggestions for each learning problem are merged.

If a learning problem only consists of positive examples (the user did not choose any articles to be excluded explicitly), negatives are generated. Therefore, instances of classes related to the examples as well as instances related via object properties are randomly picked as negative examples. They help the learning algorithm to determine, which characteristics of the examples are outstanding, e.g. it is probably not interesting that Alber Einstein has some birthdate, but it is interesting that this person won an important prize (the noble prize). Furthermore, when collecting the necessary background knowledge, certain filters are applied, that assure adequate results, e.g. we filter out the SKOS relationships in DBpedia, but keep the YAGO hierarchy, because it is better suited for our approach. We also perform methods to ensure that the extracted fragment is in OWL DL, which we do not want to describe in detail here due to lack of space.

Finally, the learned class descriptions are converted to natural language (in order to be readable by end users). URIs are replaced by labels and the class description constructors are replaced by human readable strings. We configured the algorithm to use only a sublanguage of OWL by exluding negation and universal quantification. This allows for a simple natural language conversion.

4 Conclusions and Future Work

We presented a user interface for browsing and searching in DBpedia and interlinked data sets. In particular, we introduced navigation suggestions as a means to support the user, which heavily relies on the semantics of underlying data. We also believe that this is the first time that supervised symbolic Machine Learning is routinely used in an application targeted for end users.

Being a prototype, there are a few areas, where DBpedia Navigator can be improved: 1. Large parts of the schema relevant for the learning algorithm (class/property hierarchy, domain/ranges of properties, disjoint classes) could be kept in memory, while the fragment selection algorithm is modified to work on the SPARQL endpoint only for those resources not in memory. Currently, the relevant schema portion is recreated on each request by querying the SPARQL endpoint, which causes considerable delays until the user can view the navigation suggestions. 2. The learning algorithm, which is currently suitable for learning a large fraction of available OWL constructs can be tailored to a simpler language, specifically extensions of EL. Apart from that, as usual, work needs to be done to transform the prototype into an end user application.

References

1. Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. DBpedia: A nucleus for a web of open data. In *ISWC/ASWC (2007)*, LNCS (4825), pages 722–735. Springer, 2007.
2. Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
3. Sebastian Hellmann, Jens Lehmann, and Sören Auer. Learning of owl class descriptions on very large knowledge bases. In *7th International Semantic Web Conference (ISWC)*, 2008.
4. Jens Lehmann and Pascal Hitzler. Foundations of refinement operators for description logics. In *17th Int. Conf. on Inductive Logic Programming (ILP)*, 2007.
5. Jens Lehmann and Pascal Hitzler. A refinement operator based learning algorithm for the ALC description logic. In *Proceedings of the 17th International Conference on Inductive Logic Programming (ILP)*, volume 4894 of *Lecture Notes in Computer Science*, pages 147–160. Springer, 2008.
6. Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *WWW*, pages 697–706. ACM, 2007.