Universal OWL Axiom Enrichment for Large Knowledge Bases

Lorenz Bühmann and Jens Lehmann

Universität Leipzig, Institut für Informatik, AKSW,
Postfach 100920, D-04009 Leipzig, Germany,
{buehmann|lehmann}@informatik.uni-leipzig.de
http://aksw.org

Abstract. The Semantic Web has seen a rise in the availability and usage of knowledge bases over the past years, in particular in the Linked Open Data initiative. Despite this growth, there is still a lack of knowledge bases that consist of high quality schema information and instance data adhering to this schema. Several knowledge bases only consist of schema information, while others are, to a large extent, a mere collection of facts without a clear structure. The combination of rich schema and instance data would allow powerful reasoning, consistency checking, and improved querying possibilities as well as provide more generic ways to interact with the underlying data. In this article, we present a light-weight method to enrich knowledge bases accessible via SPARQL endpoints with almost all types of OWL 2 axioms. This allows to semi-automatically create schemata, which we evaluate and discuss using DBpedia.

1 Introduction and Motivation

The Semantic Web has recently seen a rise in the availability and usage of knowledge bases, as can be observed within the DataHub[1] and other repositories. Despite this growth, there is still a lack of knowledge bases that consist of sophisticated schema information and instance data adhering to this schema. Several knowledge bases, e.g. in the life sciences, only consist of schema information, while others are, to a large extent, a collection of facts without a clear structure, e.g. information extracted from databases. The combination of sophisticated schema and instance data would allow powerful reasoning, consistency checking, and improved querying. Schema enrichment, as described in this article, allows to create schemata base based on existing data.[2]

Example 1. As an example, consider a knowledge base containing a property `birthPlace` and subjects in triples of this property, e.g. Brad Pitt, Angela Merkel, Albert Einstein etc. Our enrichment algorithms could, then, suggest

[1] http://thedatahub.org/

[2] The approach of not creating schema upfront is sometimes referred to as "grass roots" approach or "after the fact" schema creation.

that the property `birthPlace` may be functional and has the domain `Person` as it is encoded via the following axioms in Manchester OWL syntax[3]:

```
ObjectProperty: birthPlace
    Characteristics: Functional
    Domain: Person
    Range: Place
    SubPropertyOf: hasBeenAt
```

Adding such axiom to a knowledge base can have several benefits: 1.) The axioms serve as documentation for the purpose and correct usage of schema elements. 2.) They improve the application of schema debugging techniques. For instance, after adding the above axioms the knowledge base would become inconsistent if a person has two different birth places due to the functionality axiom. Specifically for the DBpedia knowledge base, we observed an error in which a person was asserted to be born in Lacrosse, the game, instead of Lacross, the city in the United States. Such errors can be automatically detected when schema information such as the range restriction is present (assuming disjointness of the classes `Place` and `Game`). 3.) Additional implicit information can be inferred, e.g. in the above example the birth place of a person can be inferred to be one of the places a person has stayed at. The main purpose of our research is to reduce the effort of creating and maintaining such schema information.

We implemented our enrichment methods in the DL-Learner[4] framework [15] based on earlier work in [11,22,19] and the ORE tool [18] [5] contains a graphical interface for them. Whereas previously we focused on equivalence and subclass axioms, we describe how to support a broader range of OWL axioms in this article. In particular, we advance the current state of the art as follows:

- support for suggesting the following axioms to enrich a knowledge base:
  - class and property hierarchy (subsumption, equivalence, disjointness)
  - property characteristics (transitivity, (a)symmetry, (inverse)functionality, (ir)reflexivity)
  - inverse properties
- support for knowledge bases accessible via SPARQL endpoints
- scalability of algorithms via sampling
- DL-Learner command line interface and ORE web interface for the algorithms are available as open source

The article is structured as follows: we briefly described the term schema enrichment and give an overview of existing approaches in Section 2. The enrichment approach itself is described in Section 3. To be able to separate the process of generating enrichments from the process of manual supervision by a knowledge engineer, we need to store the suggestions. We do this via an ontology, which is described in Section 4. We then continue by giving some preliminary evaluation results for applying the algorithms on DBpedia in Section 5. Finally, we conclude and describe future work.

[3] For details on Manchester OWL syntax (e.g. used in Protégé, OntoWiki) see `http://www.w3.org/TR/owl2-manchester-syntax/`.

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

[5] `http://ore-tool.net`

2 Knowledge Base Enrichment Overview

The term *enrichment* in this article refers to the extension of a knowledge base schema. It describes the process of increasing the expressiveness and semantic richness of a knowledge base. Enrichment methods can typically be applied in a *grass-roots* approach to knowledge base creation. In such an approach, the whole ontological structure is not created upfront[6], but evolves with the data in a knowledge base. Ideally, this enables a more agile development of knowledge bases, which could become an interesting alternative to more traditional ontology engineering methods.

Knowledge base enrichment can be seen as a sub-discipline of ontology learning. Ontology learning is more general in that it can rely on external sources, e.g. written text, to create an ontology. The term knowledge base enrichment is typically used when already existing data in the knowledge base itself is analysed to improve its schema. Enrichment methods span several research areas like knowledge representation and reasoning, machine learning, statistics, natural language processing, formal concept analysis and game playing. Ontology enrichment usually involves applying heuristics or machine learning techniques to find axioms, which can be added to an existing ontology. Naturally, different techniques have been applied depending on the specific type of axiom.

One of the most complex tasks in ontology enrichment is to find *definitions* of classes. This is strongly related to Inductive Logic Programming (ILP) [26] and more specifically supervised learning in description logics. Research in this area is not purely focused on ontology enrichment, but has other applications, e.g. drug efficiency prediction in the life sciences. Work on learning in description logics goes back to e.g. [6,7], which used so-called *least common subsumers*. Later, [4] invented a refinement operator for $\mathcal{ALER}$ and proposed to solve the problem by using a top-down approach. [8,12,13] combine both techniques and implement them in the YINYANG tool. However, those algorithms tend to produce long and hard-to-understand expressions. The algorithms implemented in DL-Learner [20,21,14,22] overcome this problem and investigate the learning problem and the use of top down refinement in detail. DL-FOIL [9] is a similar approach, which is based on a mixture of upward and downward refinement of class expressions. They use alternative measures in their evaluation, which take the open world assumption into account, which was not done in ILP previously. Most recently, CELOE [16] implements appropriate heuristics and adaptations for learning definitions in ontologies. We use this algorithm for learning definitions, but go beyond it by including support for many different axiom types.

A different approach to learning the definition of a named class is to compute the so called *most specific concept* (msc) for all instances of the class. The most specific concept of an individual is the most specific class expression, such that the individual is instance of the expression. One can then compute the *least common subsumer* (lcs) [3] of those expressions to obtain a description of the named class. However, in expressive description logics, an msc does not need to exist and the lcs is simply the disjunction of all expressions. For light-weight logics, such as $\mathcal{EL}$, the approach appears to be promising. Other approaches, e.g. [23] focus

[6] Talk by Tim Berners-Lee which advocates to get "raw data now": `http://www.ted.com/talks/tim_berners_lee_on_the_next_web.html`

on learning in hybrid knowledge bases combining ontologies and *rules*. Usually, hybrid approaches are a generalisation of concept learning methods, which enable powerful rules at the cost of efficiency (because of the larger search space). Similar as in knowledge representation, the tradeoff between expressiveness of the target language and efficiency of learning algorithms is a critical choice in symbolic machine learning.

Another enrichment task is *knowledge base completion*. The goal of such a task is to make the knowledge base complete in a particular well-defined sense. For instance, a goal could be to ensure that all subclass relationships between named classes can be inferred. The line of work starting in [28] and further pursued in e.g. [2] investigates the use of *formal concept analysis* for completing knowledge bases. It is promising, although it may not be able to handle noise as well as a machine learning technique. A Protégé plugin [29] is available. [32] proposes to improve knowledge bases through relational exploration and implemented it in the *RELExO framework*[7]. It focuses on simple relationships and the knowledge engineer is asked a series of questions. The knowledge engineer either must positively answer the question or provide a counterexample.

[33] focuses on learning *disjointness* between classes in an ontology to allow for more powerful reasoning and consistency checking. To achieve this, it can use the ontology itself, but also texts, e.g. Wikipedia articles corresponding to a concept. One of the closely related and most recent work in the area is statistical schema induction via *association rule mining* [31]. Association rules are a form of implication patterns and used to discover regularities in a data set. For instance, in [31] an association rule $A \implies B$ with sufficiently high confidence and support between two classes A and B indicates that introducing a subclass relationship $A \sqsubseteq B$ may be appropriate.

Another type of ontology enrichment is schema mapping. This task has been widely studied and will not be discussed in depth here. Instead, we refer to [5] for a survey on ontology mapping. Schema mapping is not integrated in the presented prototype.

Type/Aim	**References**
Taxonomies	[34,31]
Definitions	ILP approaches: [20,21,22,16,9,8,12,13,4], genetic approaches: [14]
Super Class Axioms	[16,31]
Rules in Ontologies	[23,24]
Disjointness	[33]
Property Chains	[31]
Alignment	challenges: [30], recent survey: [5]
Completion	formal concept analysis and relational exploration [2,32,29]

Table 1. Work in ontology enrichment grouped by type or aim of learned structures.

[7] `http://code.google.com/p/relexo/`

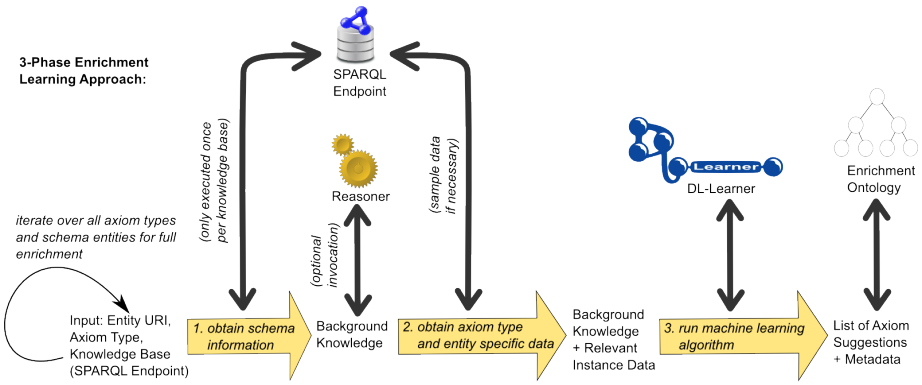

Fig. 1. 3-Phase Enrichment Workflow

3 Enrichment with OWL Axioms

There is a large variety of axiom types in OWL, which we support in our enrichment tool. We first describe our general methodology for creating enrichment suggestions and then present details for each axiom type in separate sections.

3.1 General Method

In this part, we will describe the light-weight learning methods for obtaining enrichment suggestions. The methods usually take an entity (a class or property in our case) as input, generates a set of OWL axioms as output and proceeds in three phases (see Figure 1):

1. In the first phase, SPARQL queries are used to obtain general information about the knowledge base, in particular we retrieve axioms, which allow to construct the class hierarchy. It can be configured whether to use an OWL reasoner for inferencing over the schema or just taking explicit knowledge into account.[8] Naturally, the schema only needs to be obtained once and can then be re-used by all algorithms and all entities.
2. The second phase consists of obtaining data via SPARQL, which is relevant for learning the considered axiom. We will briefly describe this phase for each axiom type in the following sections.
3. In the third phase, the score of axiom candidates is computed and the results returned.

Many of our employed heuristics to suggest axioms are based on counting. For instance, when determining whether a class A is appropriate as domain of a property p, we count the number of triples using p in predicate position and the number of subjects in those triples which are instances of A. The latter value is divided by the first value to obtain a score. We illustrate this using a simple example. Let the following triples be given (Turtle syntax):

[8] Note that the OWL reasoner only loads the schema of the knowledge base and, therefore, this option usually works even in cases with several hundred thousand classes in our experiments, which used the HermiT reasoner.

```
@prefix dbpedia: <http://dbpedia.org/resource/>.
@prefix dbo:     <http://dbpedia.org/ontology/>.
dbpedia:Luxembourg dbo:currency     dbpedia:Euro;
                   rdf:type         dbo:Country.
dbpedia:Ecuador    dbo:currency     dbpedia:US_dollar;
                   rdf:type         dbo:Country.
dbpedia:Ifni       dbo:currency     dbpedia:Spanish_peseta;
                   rdf:type         dbo:PopulatedPlace.
dbo:Country        rdfs:subClassOf  dbo:PopulatedPlace.
```

In the above example, we would obtain a score of 66,7% (2 out of 3) for the class `dbo:Country` and 100% (3 out of 3) for the class `dbo:PopulatedPlace`[9] as candidates for the range of the property `dbo:currency`.

A disadvantage of using this straightforward method of obtaining a score is that it does not take the *support* for an axiom in the knowledge base into account. Specifically, there would be no difference between having 100 out of 100 correct observations or 3 out of 3 correct observations.

For this reason, we do not just consider the count, but the average of the 95% confidence interval of the count. This confidence interval can be computed efficiently by using the improved Wald method defined in [1]. Assume we have m observations out of which s were successful, then the approximation of the 95% confidence interval is as follows:

$$\max(0, p' - 1.96 \cdot \sqrt{\frac{p' \cdot (1-p')}{m+4}}) \text{ to } \min(1, p' + 1.96 \cdot \sqrt{\frac{p' \cdot (1-p')}{m+4}})$$

$$\text{with } p' = \frac{s+2}{m+4}$$

This formula is easy to compute and has been shown to be accurate in [1].

In the above case, this would change the score to 57.3% (previously 66,7%) for `dbo:Country` and 69.1% (previously 100%) for `dbo:PopulatedPlace`. This indicates that there is not much support for either of those choices in the knowledge base. 100 out of 100 correct observations would score much higher (97.8%). The actual scores for the DBpedia Live as of May 2012 are 99.1% for the class `dbo:PopulatedPlace` and 97.6% for `dbo:Country`.

Note that in this implementation, more general classes in the hierarchy would always score higher. It might be desirable to correct this by slightly penalising very general classes. The drawback of such a penalty could be that the scores would be more difficult to understand for users. We leave the decision on such a penalty and a possible implementation as an area for future work.

3.2 Learning Subclass Axioms

In this section and the following sections, we will just focus on phase 2 of the above described workflow. This phase consists of obtaining the data required for generating enrichment suggestions. Since we mainly expect the data to be available in triple stores, the data acquisition is implemented via SPARQL queries. We will briefly present the necessary SPARQL query (or queries) here.

[9] If the reasoning option is turned off, the score would be 33,3%.

The first axiom type, we consider, are subclass axioms. Generating suggestions for subclass axioms allows to create a taxonomy from instance data. Basically the data for this can be fetched in 2 different ways:

Single Query

```
SELECT ?type (COUNT(?ind) AS ?count) WHERE {
  ?ind a <$class>.
  ?ind a ?type.
} GROUP BY ?type
```

The query assumes a `$class` as input for which we want to learn superclasses. It retrieves all instances of a class and then counts the types for each of those instances. A higher count indicates better candidates for superclasses. The disadvantage of this approach is that it puts high computational load on the SPARQL endpoint in case of very large data sets. An alternative implementation is to iterate through all results as shown below.[10] This way, each individual query is inexpensive for the endpoint as the information is obtained in small chunks. Moreover, in DL-Learner, we impose runtime limits on algorithms. This means that we stop iterating through results once a configurable time threshold has been reached. The score for suggestions is then approximated from the obtained sample. The drawback of this method is that the score can only be computed on a subset of the knowledge base whereas in the first method the whole data set is taken into account.

Iterative Query

```
SELECT ?ind ?type WHERE {
  ?ind a <$class>.
  ?ind a ?type.
}
LIMIT $limit OFFSET $offset
```

3.3 Learning Disjointness

For disjointness, we can use the same query as above:

```
SELECT ?type (COUNT(?ind) AS ?count) WHERE {
  ?ind a <$class>.
  ?ind a ?type.
} GROUP BY ?type
```

The only difference in terms of the query is that this time, a lower count indicates a candidate for disjointness. When running enrichment in batch mode, the number of suggested disjointness axioms is minimsed by moving disjointness as far up the class hierarchy as possible (see Section 3.8).

In addition, we draw on [33,10] for computing disjointness. Several criteria, specifically taxonomic overlap, existing subsumption axioms and semantic similarity are used in order to determine the most useful disjointness axioms.

3.4 Property Subsumption/Disjointness

For properties, learning subsumption and disjointness is analogous to learning this kind of axioms for classes. The difference is that we count how often subject

[10] Correct pagination in SPARQL with `LIMIT` and `OFFSET` only works with the sorting of the results by using `ORDER BY`, but we omit this in this paper for simplicity.

`?s` and object `?o` in the triples for a given property `$property` are also related via other properties `?p`.

```
SELECT ?p (COUNT(?s) AS ?count) WHERE {
  ?s ?p ?o.
  ?s <$property> ?o.
} GROUP BY ?p
```

3.5 Property Domain and Range

For domains of object properties and data properties we count the occurrences of types in the subject position of triples having the property.

```
SELECT ?type COUNT(DISTINCT ?ind) WHERE {
  ?ind <$property> ?o.
  ?ind a ?type.
} GROUP BY ?type
```

For property ranges, we issue different queries depending on whether a resource is a data or object property.

Object Properties The object property case is analogous to learning domains as shown above, except this time we pay attention to the triple objects.

```
SELECT ?type (COUNT(DISTINCT ?ind) AS ?cnt) WHERE {
  ?s <$property> ?ind.
  ?ind a ?type.
} GROUP BY ?type
```

Data Properties For data properties, we make use of the fact that every triple is annotated with its datatype in RDF, i.e. we can just count occurring datatypes.

```
SELECT ?datatype COUNT(DISTINCT ?ind) WHERE {
  ?ind <$property> ?val.
} GROUP BY (DATATYPE(?val) AS ?datatype)
```

3.6 Inverse Properties

To generate axioms which state that a property `p1` is the inverse of a property `p2` we run the query below, which retrieves properties `p` having subject and object occurring in the triples for the given property `$property` in swapped positions and count how often this happens.

```
SELECT ?p (COUNT(*) AS ?cnt) WHERE {
  ?s <$property> ?o.
  ?o ?p ?s.
} GROUP BY ?p
```

3.7 Property Characteristics

Based on the axiom type which shall be learned for a given property `$property`, either the number of triples (symmetry, asymmetry), the number of distinct subjects (functionality, reflexivity, irreflexivity) or the number of distinct objects (inverse-functionality) is computed in a first step. This value is then combined with the result of the corresponding query in Table 2 .

Functionality	`SELECT COUNT(DISTINCT ?s) AS ?functional WHERE { ?s <$property> ?o1 . FILTER NOT EXISTS {?s <$property> ?o2 . FILTER(?o1 != ?o2)} }`
Inverse-Functionality	`SELECT COUNT(DISTINCT ?o) AS ?inversefunctional WHERE { ?s1 <$property> ?o . FILTER NOT EXISTS {?s2 <$property> ?o . FILTER(?s1 != ?s2)} }`
Symmetry	`SELECT (COUNT(*) AS ?symmetric) WHERE { ?s <$property> ?o . ?o <$property> ?s . }`
Asymmetry	`SELECT (COUNT(*) AS ?asymmetric) WHERE { ?s <$property> ?o . FILTER NOT EXISTS {?o <$property> ?s.} }`
Reflexivity	`SELECT (COUNT(DISTINCT ?s) AS ?reflexive) WHERE { ?s <$property> ?s . }`
Irreflexivity	`SELECT (COUNT(DISTINCT ?s) AS ?irreflexive) WHERE { ?s <$property> ?o . FILTER NOT EXISTS {?s <$property> ?s.} }`
Transitivity	`SELECT (COUNT(*) AS ?transitive) WHERE { ?s <$property> ?o . ?o <$property> ?o1 . ?s <$property> ?o1 . }`

Table 2. SPARQL queries used to learn the different types of OWL 2 property characteristics.

3.8 Batch Mode

While the described methodology is designed to be applicable for learning axioms involving specific classes or properties, it is also possible to run the enrichment script in batch mode. In this case, the script first detects all schema entities, loops over them and calls the learning methods for all axiom types. Our evaluation shows that the approach still scales to large knowledge bases, e.g. DBpedia. To prevent flooding of the SPARQL endpoints, the batch mode contains configurable options to delay the execution of successive queries, and for the case that a timeout occurs to rerun the query after some waiting period.

Running the algorithms batched allows to add further optimisations, e.g. for disjointness between classes we give the opportunity to restrict the returned suggestions to pairs of most general classes, as due to inference disjointness is propagated to lower subclasses. Further possibilities we will investigate in the future are (1) the problem of coherency, i.e. some axiom types especially disjointness combined with e.g. subsumption can lead to unsatisfiable entities, (2) iterative creation of the knowledge base, i.e. taking into account earlier learned axioms for the generation of other axiom types, and (3) the minimization of the resulting ontology, i.e. finding an ontology in which none of the axioms can be inferred from other existing axioms.

4 Enrichment Ontology

As previously discussed, enrichment is usually a semi-automatic process. Each enrichment suggestion generated by an algorithm should be reviewed by a knowledge engineer who can then decide to accept or reject it. Because of this, there is a need for serialising enrichment suggestions such that the generation of them

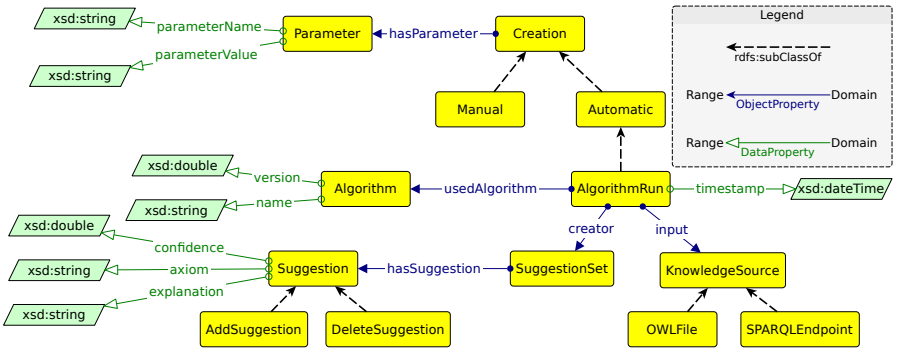

Fig. 2. Enrichment ontology used to store additional information occuring during learning proocess.

is independent of the process of accepting or rejecting them. Since all enrichment suggestions are OWL axioms, they could simply be written in an RDF or OWL file. However, this might be insufficient, because we lose a lot of metadata this way, which could be relevant for the knowledge engineer. For instance, algorithms may be able to store confidence values, statistics or even textual descriptions on why an enrichment is suggested. For this reason, we created an enrichment ontology[11], which is partially building on related efforts in [27] and `http://vocab.org/changeset/schema.html`. Such an interchange format is also relevant, because the process of generating enrichments for all schema elements in very large knowledge bases will often take several hours to complete. Furthermore, the metadata also simplifies reproducing algorithms results by storing algorithm versions, the used knowledge sources and time information. An overview of the ontology can be found in Figure 2.

5 Preliminary Evaluation

To assess the feasibility of our approaches, we evaluated them on DBpedia [17]. We performed an enrichment on the DBpedia Live knowledge base [25], which at that time consisted of 385 million triples, 3.64 million things, 272 classes, 629 object properties and 706 data properties. We used a confidence threshold of 0.7 for the algorithm runs and showed at most 10 suggestions per entity and axiom type. Table 3 contains basic runtime information on the algorithms. It shows how many enrichment suggestions were made per axiom type, the runtime of the algorithm, the average score and the average of the maximum scores of each algorithm run. The algorithms require 10-161 seconds per ontology entity. To the best of our knowledge, there are no other approaches performing the same task to which we can compare our method in general. For a small number of the reported axiom types [31] performs a similar approach using association rule mining, which yields very similar results to our approach due to high similarity of the underlying heuristics for this axiom types.

[11] Available at http://dl-learner.org/ontologies/enrichment.owl.

algorithm	Avg. nr. of #suggestions	Avg. runtime in ms	timeout in %	Avg. score	Avg. max. score
disjoint classes	10.0	11957.0	0.00	1.00	1.00
subclass	2.5	98233.0	0.00	0.95	0.98
disjoint objectproperty	10.0	12384.0	0.16	1.00	1.00
equivivalent objectproperty	1.1	12509.0	0.16	0.96	0.96
functional objectproperty	1.0	19990.0	0.48	0.89	0.89
inv.funct. objectproperty	1.0	113590.0	4.29	0.86	0.86
objectproperty domain	3.1	11577.0	0.00	0.93	0.96
objectproperty range	2.6	14253.0	0.16	0.84	0.87
objectproperty subPropertyOf	1.1	56363.0	0.32	0.93	0.93
symmetric objectproperty	1.0	12730.0	0.32	0.80	0.80
transitive objectproperty	1.0	16830.0	1.91	0.84	0.84
irreflexive objectproperty	1.0	17357.0	0.16	0.97	0.97
reflexive objectproperty	0.0	10013.0	0.16	-	-
disjoint dataproperty	10.0	137204.0	3.97	1.00	1.00
equiv. dataproperty	1.0	161229.0	4.25	0.92	0.92
funct. dataproperty	1.0	18281.0	0.42	0.94	0.94
dataproperty domain	2.8	10340.0	0.28	0.94	0.96
dataproperty range	1.0	13516.0	0.42	0.96	0.96
dataproperty subPropertyOf	1.0	91284.0	4.11	0.88	0.88
OVERALL	3.0	55389.0	1.08	0.92	0.93

Table 3. Basic information on runtime and number of suggestions of the algorithms.

Table 4 shows our evaluation results. In this evaluation we defined recall with respect to the existing DBpedia ontology. For instance, 180/185 in the subClassOf row indicates that we were able to re-learn 180 out of 185 such subclass relationships from the original DBpedia ontology. Higher numbers are an indicator that the methods do not miss many possible enrichment suggestions. The next column shows how many additional axiom were suggested, i.e. how many axioms were suggested which are not in the original DBpedia ontology. The last three columns are the result of a manual evaluation. Both authors independently observed at most 100 axioms per type (possibly less, in case fewer than 100 suggestions were made) and evaluated them manually. Three different categories were used: "yes" indicates that it is likely that they would be accepted by a knowledge engineer, "maybe" are corner cases and "no" are those, which would probably be rejected. The evaluation at this stage is preliminary as it was only conducted by the authors. A full evaluation including several datasets and external reviewers is scheduled as future work.

In summary, we observed that axioms regarding the class hierarchy basically seem to be more easy to learn than axioms building the property hierarchy. We also noticed that we could suggest new axioms for all axiom types except for the ReflexiveObjectProperty ones. The reason is that DBpedia does not contain corresponding instance data. The low recall for the range axioms of object and

axiom type	recall	additional axioms	Estimated precision		
			no	maybe	yes
SubClassOf	180/185	155	5	20	75
EquivalentClasses	0/0	1812	20	30	50
DisjointClasses	0/0	2449	0	0	100
SubObjectPropertyOf	0/0	45	18	9	18
EquivalentObjectProperties	0/0	40	40	0	0
DisjointObjectProperties	0/0	5670	0	0	100
ObjectPropertyDomain	385/449	675	10	22	68
ObjectPropertyRange	173/435	427	4	59	37
TransitiveObjectProperty	0/0	12	5	5	2
FunctionalObjectProperty	0/0	352	8	18	74
InverseFunctionalObjectProperty	0/0	173	72	3	25
SymmetricObjectProperty	0/0	3	0	0	3
ReflexiveObjectProperty	0/0	0	-	-	-
IrreflexiveObjectProperty	0/0	536	1	0	99
SubDataPropertyOf	0/0	197	86	8	6
EquivalentDataProperties	0/0	213	20	9	71
DisjointDataProperties	0/0	62	0	0	100
DataPropertyDomain	448/493	623	27	33	40
DataPropertyRange	118/597	79	0	0	100
FunctionalDataProperty	14/14	509	4	17	79

Table 4. Evaluation results.

data properties is mostly due to either missing or different type information on the triples' objects.

Below, we list some of the observations we made:

- The test set for irreflexive properties contained `dbo:isPartOf`, which is usually considered as a reflexive property. It is the only incorrect suggestion in this test set.
- The 5 missing subclass axioms are as follows:
  1. `dbo:Ginkgo subClassOf: dbo:Plant`
  2. `dbo:MixedMartialArtsLeague subClassOf: dbo:SportsLeague`
  3. `dbo:VoiceActor subClassOf: dbo:Actor` (each of those 3 axioms has only 1 triple and therefore too low support)
  4. `dbo:PoloLeague subClassOf: dbo:SportsLeague` (only 3 triples, therefore it had low support)
  5. `dbo:Bridge subClassOf: dbo:Building` (none of the bridges is actually a building according to DBpedia Live)
- As an example for the imperferct recall on object property domains, the results of the learning procedure for `dbo:hometown` are as follows: For the existing domain `dbo:Person` we only got a score of 0.3, whereas `dbo:Band` and `dbo:Organisation` achieved a score of approx. 0.7. This is because each `dbo:Band` was also a `dbo:Person` at this time in DBpedia Live.
- We discovered 3 symmetric object properties, namely `dbo:neighboringMunicipality`, `dbo:sisterCollege` and `dbo:currentPartner`.

- For most of the data properties, the learned axioms of the types SubDataPropertyOf and EquivalentDataProperties contained properties of the DBpedia namespace `http://dbpedia.org/property/`(dbp), e.g. EquivalentDataProperties(`dbo:drugbank`,`dbp:drugbank`). Mixing the two different ontologies is usually not desirable, hence the low precision in some of these cases. As a result, we now support more fine-grained control over the used ontology namespaces to avoid those problems.
- We missed some DataPropertyRanges, because sometimes the defined range in the ontology is a different datatype, compared to the one of the literal values in the triples. For instance `dbo:background` has a defined range `xsd:string`, but in the instance data the literals only have a language tag (which makes them implicit to `rdf:PlainLiteral`). `dbo:budget` (range in ontology: `xsd:double`, but `http://dbpedia.org/datatype/usDollar` used in the actual literals) is a different example. Clearly, in those cases data errors cause problems and our enrichment tool can be used to detect those.
- In some cases we learned a different datatype, so we missed the existing one and found an additional one. Most of these additional axioms would also be a reasonable but not optimal choice, e.g. for `dbo:populationTotal` we learned `xsd:integer`, whereas in the ontology `xsd:nonNegativeInteger` is defined.

6 Conclusion

We presented a set of approaches for schema enrichment, which covers most OWL 2 axioms. Those approaches were implemented and released in the DL-Learner machine learning framework. In our preliminary evaluation, we showed the feasibility of the methods for knowledge bases of the size of DBpedia.

In future work, we will investigate enhancements of the presented methods as indicated in the discussions of the respective approaches. In particular, we closely collaborate with the authors of [31] to fine-tune the approach. One of the next steps will be to investigate how to generate a coherent ontology when combining all suggestions and how to use such an ontology for debugging large knowledge bases.

References

1. Alan Agresti and Brent A. Coull. Approximate is better than "exact" for interval estimation of binomial proportions. *The American Statistician*, 52(2):119–126, 1998.
2. Franz Baader, Bernhard Ganter, Ulrike Sattler, and Baris Sertkaya. Completing description logic knowledge bases using formal concept analysis. In *IJCAI 2007*. AAAI Press, 2007.
3. Franz Baader, Baris Sertkaya, and Anni-Yasmin Turhan. Computing the least common subsumer w.r.t. a background terminology. *J. Applied Logic*, 5(3):392–420, 2007.

4. Liviu Badea and Shan-Hwei Nienhuys-Cheng. A refinement operator for description logics. In *ILP 2000*, volume 1866 of *LNAI*, pages 40–59. Springer, 2000.
5. Namyoun Choi, Il-Yeol Song, and Hyoil Han. A survey on ontology mapping. *SIGMOD Record*, 35(3):34–41, 2006.
6. William W. Cohen, Alexander Borgida, and Haym Hirsh. Computing least common subsumers in description logics. In *AAAI 1992*, pages 754–760, 1992.
7. William W. Cohen and Haym Hirsh. Learning the CLASSIC description logic: Theoretical and experimental results. In *KR 1994*, pages 121–133. Morgan Kaufmann, 1994.
8. Floriana Esposito, Nicola Fanizzi, Luigi Iannone, Ignazio Palmisano, and Giovanni Semeraro. Knowledge-intensive induction of terminologies from metadata. In *ISWC 2004*, pages 441–455. Springer, 2004.
9. Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito. DL-FOIL concept learning in description logics. In *ILP 2008*, volume 5194 of *LNCS*, pages 107–121. Springer, 2008.
10. Daniel Fleischhacker and Johanna Völker. Inductive learning of disjointness axioms. In Robert Meersman, Tharam S. Dillon, Pilar Herrero, Akhil Kumar, Manfred Reichert, Li Qing, Beng Chin Ooi, Ernesto Damiani, Douglas C. Schmidt, Jules White, Manfred Hauswirth, Pascal Hitzler, and Mukesh K. Mohania, editors, *OTM Conferences (2)*, volume 7045 of *Lecture Notes in Computer Science*, pages 680–697. Springer, 2011.
11. Sebastian Hellmann, Jens Lehmann, and Sören Auer. Learning of OWL class descriptions on very large knowledge bases. *Int. J. Semantic Web Inf. Syst.*, 5(2):25–48, 2009.
12. Luigi Iannone and Ignazio Palmisano. An algorithm based on counterfactuals for concept learning in the semantic web. In *IEA/AIE 2005*, pages 370–379, June 2005.
13. Luigi Iannone, Ignazio Palmisano, and Nicola Fanizzi. An algorithm based on counterfactuals for concept learning in the semantic web. *Applied Intelligence*, 26(2):139–159, 2007.
14. Jens Lehmann. Hybrid learning of ontology classes. In *Machine Learning and Data Mining in Pattern Recognition*, volume 4571 of *LNCS*, pages 883–898. Springer, 2007.
15. Jens Lehmann. DL-Learner: learning concepts in description logics. *Journal of Machine Learning Research (JMLR)*, 10:2639–2642, 2009.
16. Jens Lehmann, Sören Auer, Lorenz Bühmann, and Sebastian Tramp. Class expression learning for ontology engineering. *Journal of Web Semantics*, 9:71 – 81, 2011.
17. Jens Lehmann, Chris Bizer, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 7(3):154–165, 2009.
18. Jens Lehmann and Lorenz Bühmann. Ore - a tool for repairing and enriching knowledge bases. In *Proceedings of the 9th International Semantic Web Conference (ISWC2010)*, Lecture Notes in Computer Science, Berlin / Heidelberg, 2010. Springer.
19. Jens Lehmann and Christoph Haase. Ideal downward refinement in the EL description logic. In *Inductive Logic Programming, 19th International Conference, ILP 2009, Leuven, Belgium*, 2009.
20. Jens Lehmann and Pascal Hitzler. Foundations of refinement operators for description logics. In *ILP 2007*, volume 4894 of *LNCS*, pages 161–174. Springer, 2008. Best Student Paper Award.

21. Jens Lehmann and Pascal Hitzler. A refinement operator based learning algorithm for the ALC description logic. In *ILP 2007*, volume 4894 of *LNCS*, pages 147–160. Springer, 2008. Best Student Paper Award.
22. Jens Lehmann and Pascal Hitzler. Concept learning in description logics using refinement operators. *Machine Learning journal*, 78(1-2):203–250, 2010.
23. Francesca A. Lisi. Building rules on top of ontologies for the semantic web with inductive logic programming. *Theory and Practice of Logic Programming*, 8(3):271–300, 2008.
24. Francesca A. Lisi and Floriana Esposito. Learning SHIQ+log rules for ontology evolution. In *SWAP 2008*, volume 426 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
25. Mohamed Morsey, Jens Lehmann, Sören Auer, Claus Stadler, , and Sebastian Hellmann. Dbpedia and the live extraction of structured data from wikipedia. *Program: electronic library and information systems*, 46:27, 2012.
26. Shan-Hwei Nienhuys-Cheng and Ronald de Wolf, editors. *Foundations of Inductive Logic Programming*, volume 1228 of *LNCS*. Springer, 1997.
27. Raúl Palma, Peter Haase, Óscar Corcho, and Asunción Gómez-Pérez. Change representation for OWL 2 ontologies. In Rinke Hoekstra and Peter F. Patel-Schneider, editors, *OWLED*, volume 529 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.
28. Sebastian Rudolph. Exploring relational structures via FLE. In Karl Erich Wolff, Heather D. Pfeiffer, and Harry S. Delugach, editors, *ICCS 2004*, volume 3127 of *LNCS*, pages 196–212. Springer, 2004.
29. Baris Sertkaya. OntocomP system description. In Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler, editors, *Proceedings of the 22nd International Workshop on Description Logics (DL 2009), Oxford, UK, July 27-30, 2009*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.
30. Pavel Shvaiko and Jerome Euzenat. Ten challenges for ontology matching. Technical report, August 01 2008.
31. Johanna Völker and Mathias Niepert. Statistical schema induction. In Grigoris Antoniou, Marko Grobelnik, Elena Paslaru Bontas Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Z. Pan, editors, *ESWC (1)*, volume 6643 of *Lecture Notes in Computer Science*, pages 124–138. Springer, 2011.
32. Johanna Völker and Sebastian Rudolph. Fostering web intelligence by semi-automatic OWL ontology refinement. In *Web Intelligence*, pages 454–460. IEEE, 2008.
33. Johanna Völker, Denny Vrandecic, York Sure, and Andreas Hotho. Learning disjointness. In *ESWC 2007*, volume 4519 of *LNCS*, pages 175–189. Springer, 2007.
34. Harris Wu, Mohammad Zubair, and Kurt Maly. Harvesting social knowledge from folksonomies. In *Proceedings of the seventeenth conference on Hypertext and hypermedia*, HYPERTEXT '06, pages 111–114, New York, NY, USA, 2006. ACM.