DL-Learner - A Framework for Inductive Learning on the Semantic Web

Lorenz Bühmann[a], Jens Lehmann[b], Patrick Westphal[a]

[a] University of Leipzig, Institute of Computer Science, AKSW Group, Augustusplatz 10, D-04009 Leipzig, Germany
E-mail: {lastname}@informatik.uni-leipzig.de
[b] University of Bonn, Institute of Computer Science, Römerstr. 164, D-53117 Bonn, Germany
E-mail: jens.lehmann@cs.uni-bonn.de

Abstract

In this system paper, we describe the DL-Learner framework, which supports supervised machine learning using OWL and RDF for background knowledge representation. It can be beneficial in various data and schema analysis tasks with applications in different standard machine learning scenarios, e.g. in the life sciences, as well as Semantic Web specific applications such as ontology learning and enrichment. Since its creation in 2007, it has become the main OWL and RDF-based software framework for supervised structured machine learning and includes several algorithm implementations, usage examples and has applications building on top of the framework. The article gives an overview of the framework with a focus on algorithms and use cases.

Keywords: System Description, Machine Learning, Supervised Learning, Semantic Web, OWL, RDF

1. Introduction

Over the past two decades, we have witnessed a transition from an industrial driven society to a data and knowledge driven society. This trend is accompanied by a significant increase in research interest in and importance of (large-scale) data processing methods. In this broad field, semantic technologies have emerged as a means to structure, publish and integrate data. In particular, the RDF and OWL knowledge representation W3C standards are used in thousands of knowledge bases containing billions of facts.[1]

A major challenge that research faces today is to analyse this growing amount of information to obtain insights into the underlying problems. In many cases, in particular in the life sciences, it is beneficial to employ methods that are able to use the complex structure of available background knowledge when learning hypotheses. DL-Learner is an open software framework, which contains several such methods. It has the primary goal to serve as a platform for facilitating the implementation and evaluation of supervised structured machine learning methods using semantic background knowledge.

The most common scenario we consider is to have a background knowledge base in OWL and be additionally provided with sets of individuals in our knowledge base which serve as positive and negative examples. The goal is to find a logical formula, e.g. an OWL *class expression*[2], such that all/many of the positive examples are *instances* of this expression and none/few of the negative examples are instances of it. The primary purpose of learning is to find a class expression which can classify unseen individuals (i.e. not belonging to the examples) correctly. It is also important that the obtained class expression is easy to understand for a domain expert. We call these criteria *accuracy* and *readability*.

As an example, consider the problem to find out whether a chemical compound can cause cancer. In this case, the background knowledge contains information about chemical compounds in general and certain concrete compounds we are interested in. The positive examples are those compounds causing cancer, whereas the negative examples are those compounds not causing cancer. The prediction for those examples may have been obtained from experiments and expensive long-term research trials. A learning algorithm can now derive a hypothesis from examples and background knowledge, e.g. a learned class expression in natural language could be "chemical compounds containing more than three phosphorus atoms". (Of course, in practice the expression will be more complex to obtain a reasonable accuracy.) Using this class expression, we can now classify unseen chemical compounds.

To solve this and similar problems, researchers need to overcome hurdles which are highly important for machine learning in general: Algorithms have to process complex background knowledge, possibly coming from several sources. Logical inference is needed during the learning process to drive the algorithms. During their runtime, algorithms frequently need to evaluate thousands or millions of hypotheses and use the results of those tests to determine their learning strategy. The expressions which are eventually learned, can often be arbitrarily nested and in some cases need to portray complex relationships while still being as easy as possible to understand for domain experts. Some of those machine learning challenges have been identified by leading researchers, e.g. in [1] and [2], and DL-Learner aims to provide a platform to facilitate researchers in their quest for solutions.

[1] `http://lodstats.aksw.org/`

[2] `http://www.w3.org/TR/owl2-syntax/#Class_Expressions`

A previous system paper on DL-Learner appeared in 2009 in the Journal of Machine Learning Research [3]. Compared to this system description, the following major changes are:

- Framework design: The framework has been generalised from being focused on learning OWL class expressions using OWL ontologies as background knowledge to a more generic supervised structured machine learning framework. Components are integrated via Java Beans and the Java Spring framework, which allows more fine-grained and more flexible interaction between them.

- New algorithms for learning SPARQL queries (as feedback component in question answering), fuzzy description logic expressions, parallel OWL class expression learning, a special purpose algorithm for the $\mathcal{EL}$ description logic, two algorithms for knowledge base enrichment of almost all OWL 2 axioms from SPARQL endpoints as well as an algorithm combining inductive learning with natural language processing have been integrated.

- Scalability enhancements: There are now statistical sampling methods available for dealing with a large number of examples as well as different knowledge base fragment selection methods for handling large knowledge bases in general.

- Major engineering changes: In 2011, the whole framework was refactored to be more easily extensible. Moreover, algorithms are continuously extended with options based on received feature requests. In the same manner, new APIs and reasoners are continuously upgraded.

- Code repository statistics: About 3,500 commits were made which led to more than 4,500,000 changed lines of code. 30 new contributors could be acquired, 52 bugs could be fixed and 27 feature requests could be realized.

The article is structured as follows: In Section 2, we give a description of the problems DL-Learner aims to solve. Subsequently, in Section 3, the software framework is described. We summarise core algorithms implemented in DL-Learner in Section 4. Implementation statistics and notes are given in Section 5. Use cases of DL-Learner in different problem areas are covered in Section 6. In Section 7, we describe related work and give an outlook in Section 8.

2. Learning Problems

The process of learning in logics, i.e. trying to find high-level explanations for given data, is also called *inductive reasoning* as opposed to *inference* or *deductive reasoning*. Deductive reasoning is known as the process of deriving logically certain conclusions from a set of general statements that are known to be true, e.g. given a statement like "Every bird can fly", we can deductively derive that the bird "tweety" can fly. Contrarily, the concept of inductive reasoning is to construct general statements from a given set of examples. For instance, given ten ravens out of which nine have a black color, one could inductively derive that "all ravens are black" even though it does not hold for all ravens and bears some degree of uncertainty. Learning problems which are similar to the one we will analyse, have been investigated in *Inductive Logic Programming* [4].

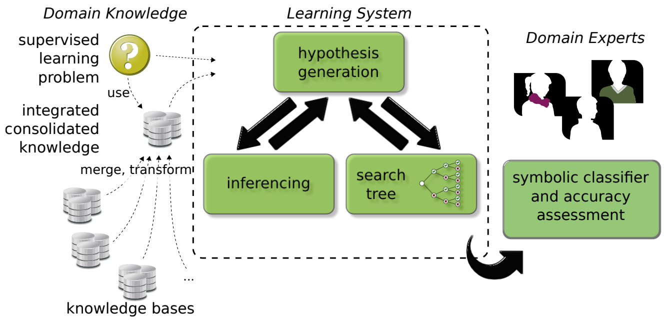


Figure 1: General learning workflow.

The goal of DL-Learner is to provide a structural framework and reusable components for solving those induction problems. Figure 1 depicts a typical workflow from a user's perspective. On the left hand side, there are several knowledge bases which together form the *background knowledge* for a given task. Within that background knowledge, some resources are selected as positive and some others as negative examples. In a medical setting, the resources could be patients reacting to a treatment (positive examples) and patients not reacting to a treatment (negative examples). Those are then processed by a supervised machine learning algorithm and return (in most cases in DL-Learner) a *symbolic classifier*. This classifier is human readable and expressed in a logical form, e.g. as a complex description logic concept or a SPARQL query. It serves two purposes: First, due to its logical representation it should give insights into the underlying problem, showing which concepts are relevant to distinguish positive and negative examples. Furthermore, the result can also be used to classify unseen resources, e.g. by checking whether they are an instance of the learned concept using an OWL reasoner, or whether they are returned by a SPARQL endpoint executing a learned query.

In DL-Learner, the following learning problems are relevant:

Standard Supervised Learning Let the name of the background ontology be $\mathcal{O}$. The goal in this learning problem is to find an OWL class expression C such that all/many positive examples are instances of C w.r.t. $\mathcal{O}$ and none/few negative examples are instances of C w.r.t. $\mathcal{O}$.

Positive Only Learning In case only positive examples are available, it is desirable to find a class expression that covers the positive examples while still generalising sufficiently well (usually measured on unlabelled data).

Class Learning In class learning, you are given an existing class A within an ontology $\mathcal{O}$ and want to describe it. This is similar to the previous problem in that you can use the instances of the class as positive examples. However, you can make use of existing knowledge about A in the ontology and (obviously) A itself should not be a solution.

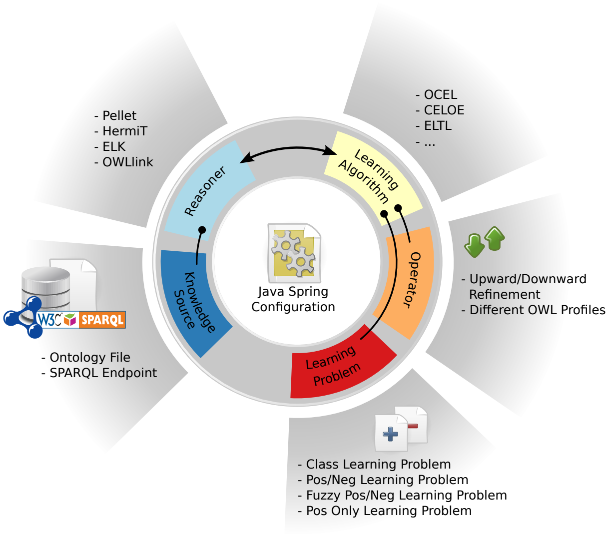


Figure 2: Overview of core DL-Learner components.

In addition, there are different nuances of the above learning problems which depend on how negative knowledge should be treated (related to the open world assumption in description logics): The problems can be treated as a binary problem (hypotheses should cover positive examples and not cover negative examples) or a ternary problem (hypotheses should cover positive examples, cover the negation of negative examples and not cover all other resources). The preferable setting partially depends on the structure of the background knowledge. For ternary learning problems, more sophisticated schema axioms, e.g. containing negation, disjunction or certain property restrictions, are required to obtain useful results. DL-Learner implements standard binary measures, e.g. predictive accuracy, F-measure, and all ternary measures described in [5]. Obviously, in most cases, we will not find a perfect solution to the learning problem, but rather an approximation, the degree of which is managed by setting suitable thresholds representing the tolerance to noise/errors, i.e. the fraction of uncovered positive resp. covered negative examples.

3. Overview of the Framework

The DL-Learner framework provides means to flexibly build concept learning algorithms. Several (Java) interfaces, adapters and external API connectors are part of the implementation. Figure 2 shows the main parts of this structure:

1. *Knowledge sources* specify where and how to retrieve data. Currently, most RDF and OWL serialisation formats are supported. Data can be retrieved locally or remotely. Retrieving data from SPARQL endpoints is also supported, including various options to extract fragments, filter and pre-process data [6] as well as several retrieval strategies differing in performance. A single learning problem can have multiple knowledge sources including mixtures of different types of sources.
2. *Reasoners* perform inference over knowledge sources. DL-Learner supports connecting reasoners via the OWL API, OWLlink as well as direct access to e.g. Pellet if advanced features not covered in the standard interfaces are needed. DL-Learner also implements own approximative reasoners (not sound and/or incomplete) for high performance hypothesis testing. Note that learning algorithms are not required to use reasoning, i.e. can also work only on the asserted knowledge, but indeed can benefit from inferred knowledge – there is a trade-off between computational complexity and expressivity.
3. *Learning problems* define the task to solve (see Sec. 2). Learning problems are typically used by learning algorithms for hypothesis testing. DL-Learner provides statistical sampling methods which allow efficient hypothesis testing even in the presence of a high number of examples [7]. Those methods approximate objective functions, such as F-measure, by iteratively sampling from the given examples until the confidence interval around the approximated objective function value is sufficiently small.
4. *Refinement Operators* are used to traverse through the space of possible hypotheses. DL-Learner implements a set of refinement operators, which can be configured towards particular fragments of OWL as well as an efficient operator for the $\mathcal{EL}$ language specifically.
5. *Learning algorithms* implement the core learning strategy. In the next section, we will summarise currently implemented algorithms and briefly point out how contributors can add further algorithms in Section 5.

4. Learning Algorithms

In early work, we provided theoretical foundations for the field on top of which we developed algorithms derived from Inductive Logic Programming and genetic programming [8]. This was extended to very expressive schemata [9] and learning problems with a lot of instance data [6]. Later, we extended the theoretical and algorithmic foundations for a) learning complex definitions in ontologies [7], b) generic schema enrichment [10, 11], c) fuzzy description logics [12], d) the light-weight $\mathcal{EL}$-description logic [13] and e) combinations of natural language processing and concept learning [14]. We will briefly describe the algorithms resulting from those lines of research.

Refinement Operator Algorithms

The first category of algorithms is based on so-called *refinement operators*. The design of those algorithms is motivated by the fact that, generally, learning can be seen as the search for a correct concept definition in an ordered space $(\Sigma, \preceq)$. In such a setting, one can define suitable operators to traverse the search space.

Definition 1 (refinement operator)**.** *Given a quasi-ordered[3] search space* $(\Sigma, \preceq)$

- *a* downward refinement operator *is a mapping* $\rho : \Sigma \rightarrow 2^\Sigma$ *such that* $\forall \alpha \in \Sigma \quad \rho(\alpha) \subseteq \{\beta \in \Sigma \mid \beta \preceq \alpha\}$

[3] A quasi-ordering is a reflexive and transitive relation.

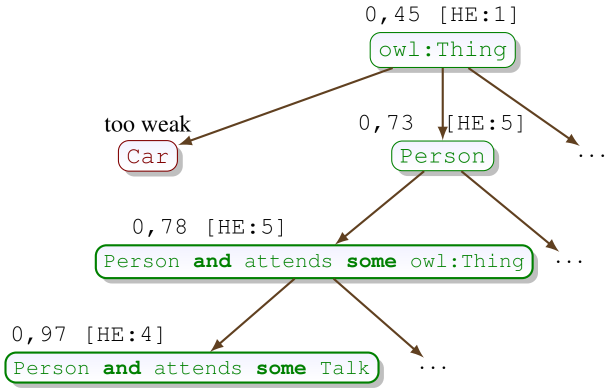


Figure 3: Search tree used in OCEL and CELOE algorithm.

- *an* upward refinement operator *is a mapping* $\delta : \Sigma \to 2^{\Sigma}$ *such that* $\forall \alpha \in \Sigma \quad \delta(\alpha) \subseteq \{\beta \in \Sigma \mid \alpha \preceq \beta\}$

Intuitively, a downward (resp. upward) refinement operator returns a set of more specific (resp. general) concepts. The goal is often to devise operators that have many useful properties like finiteness, non-redundancy, properness and completeness while still allowing to efficiently traverse throughout the search space in pursuit of good hypotheses. However, there are theoretical limitations on what properties can be combined for more expressive description logics [9], e.g. it is possible to design an operator that is finite and complete but there exists no operator that is finite, proper and complete. This is not a problem at all because some of the missing properties of an operator can be handled algorithmically, i.e. we are able to workaround for instance infinity and redundancy.

OCEL (OWL Class Expression Learner) was initially devised for learning in the description logic $\mathcal{ALC}$, but was later extended to cover other parts of OWL as well, e.g. nominals. The general idea is to use a proper and complete refinement operator to build a search tree while using heuristics which control how the search tree is traversed. The algorithm uses techniques to cope with redundancy and infinity, in particular, infinity is handled by the ability to revisit nodes in the search tree several times and perform incremental applications of the refinement operator. Figure 3 visualises a search tree of OCEL, starting from the most general concept `owl:Thing` as the root node to more specific concepts like `Person` or `Person` **`that`** `attends` **`some`** `talk`. Nodes are annotated with their score and the number of times they have been expanded (denoted by the HE value). Some nodes are too weak to eventually lead to competitive learning problem solutions, i.e. the number of uncovered positive examples is above a given threshold. They are never visited, which allows the algorithm to ignore those parts of the search space resulting in improved efficiency.

CELOE (Class Expression Learning for Ontology Engineering) [7] is an evolution of OCEL, which contains adaptions specific for the class learning problem. One of the main modifications is a different heuristic which is more biased towards short concepts. Those are more likely to be relevant in the ontology creation and maintenance scenarios as the expressions created and maintained by humans are often simpler than those required for complex prediction tasks.

ELTL (EL Tree Learner) is an algorithm optimized for learning EL trees using a refinement operator [13] designed for the OWL EL profile. This operator has been proven to be ideal, i.e. finite, proper and complete. One particular feature of the developed operator is that it integrates $\mathcal{EL}$ reasoning via so called simulations, which makes the generation of refinements very efficient.

The *ISLE* (Inductive Statistical Learning of Expressions) approach [14] is an extension of the ELTL algorithm, which can take textual evidence into account in addition to structural background knowledge. It is particularly optimised for the ontology learning use cases and assumes that the classes of the ontology to construct are described in a text corpus. The information in the corpus is used to modify the search heuristic resulting in learned expressions which are usually not more accurate with respect to available background knowledge, but have been shown to be more accurate in manual human evaluation.

OWL Schema Learning Algorithms

Another group of algorithms aims at generating OWL axioms which can be used to enrich or revise the existing schema of a knowledge base by analysing the instance data [10]. This process can be applied to all types of OWL schema entities, i.e. not only to classes but also to object and data properties. In particular, this means that we do not only support the generation of axioms regarding the class hierarchy, e.g. “every employee is a person”, but also allow for computing the domain or range of a property, whether it should be declared as functional, transitive etc. As an example, the algorithms may suggest that the property `birthDate` has the domain `Person` and the range `xsd:date`, and moreover is supposed to be functional, i.e. there is only one birth date for each person. The general workflow of schema enrichment is visualised in Figure 4. More details will be given in Sec. 6.2.

In addition to the basic axiom types of OWL, which can be semi-automatically added to the knowledge base, DL-Learner also support more complex *axiom patterns* [11]. Instead of learning general expressions, this technique is tailored to a suite of axiom types which have been determined to be relevant by analysing a large corpus of ontologies on the web (upper part of Figure 4). After retrieving the general schema information, those patterns are used in SPARQL-based pattern detection algorithms, which can be executed over large datasets stored in SPARQL endpoints (lower part of Figure 4). On top of the data retrieved by those patterns, different scoring algorithms can be applied to compute confidence values and rank suggestions. The approach optionally allows to perform schema reasoning, but does not load instance data into the reasoner for efficiency reasons. While this approach lacks generality, it is scalable and based on real schema usage patterns. This renders the approach a suitable algorithm candidate for ontology learning.

Other Algorithms

In addition to the previously shown algorithms, DL-Learner contains algorithms that have been inspired by different technologies resp. use cases:

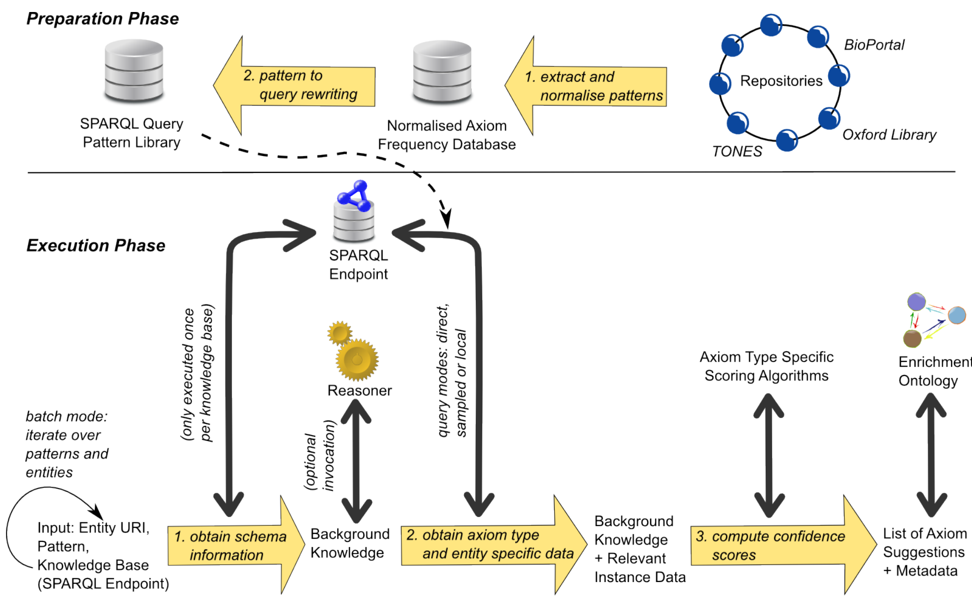
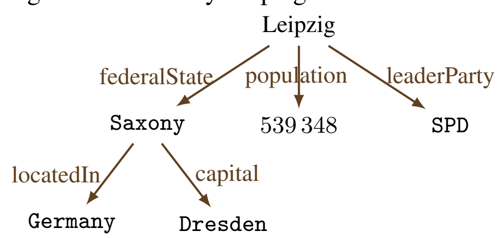
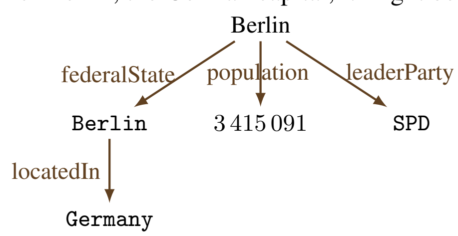


Figure 4: The general workflow of knowledge base enrichment and its pattern based extension: Frequent axiom patterns in various ontology portals are detected and converted into SPARQL query patterns (upper part). Those are then applied to other datasets to enrich them with further axioms (lower part).

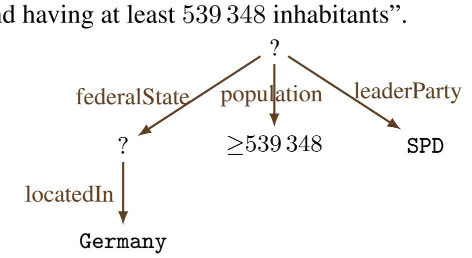
QTL is an algorithm that makes use of so-called *query trees*, which is intuitively a tree-like representation of information about a particular individual. For example, a query tree representing the German city Leipzig could be



and for Berlin, the German capital, it might be



Based on those input trees, the least general generalization (LGG) is computed, i.e. the most specific query tree that subsumes all positive examples will be returned. For the example above, the LGG could be represented by the query tree below, which can be roughly described as "German cities lead by SPD and having at least 539 348 inhabitants".



The negative examples are used for generalisation of this input tree in the QTL algorithm.

Finally, the tree or its corresponding SPARQL query will be returned:

```
SELECT DISTINCT ?s WHERE {
 ?s ont:leaderParty :SPD .
 ?s ont:federalState ?o1 .
 ?o1 ont:locatedIn :Germany .
 ?s ont:population ?o2 .
 FILTER(?o2 >= 539348) }
```

PARCEL, the parallel class expression learning algorithm [15] aims at computing partial definitions of a learning problem, which are then aggregated to complete solutions. This approach lends itself naturally to parallelization and has been applied in an abnormality detection usage scenario in smart homes, in which e.g. the learned result for the concept `normal showering` is the disjunction of the partial definitions like

```
1. activityHasDuration some (hasDurationValue some
     double[>= 4.5] and hasDurationValue some double
     [<= 15.5])
2. activityHasDuration some (hasDurationValue some
     double [>= 15.5] and hasDurationValue some
     double[<= 19.5]) and activityHasStarttime some
     {Spring}
```

Fuzzy DLL [12] is an algorithm that is targeted at vague and imprecise domains. While building on CELOE, it uses a fuzzy description logic reasoner and can optimise towards fuzzy objective functions.

Influenced by the field of *Genetic Programming*, DL-Learner contains algorithms using evolutionarily inspired methods for concept learning. Those algorithms [8] introduce new genetic operators to make better use of background knowledge than standard operators. Although those algorithms are meanwhile usually outperformed by newer approaches, they can still be interesting in case there are not many schema restrictions (e.g. disjunctions, domain and range restrictions) as those are better exploited by CELOE and OCEL.

5. Implementation

DL-Learner is developed under the GPL 3 license and primarily written in Java. The project was started in 2007 and had 30 developers contributing since then with 4 researchers in the core development team currently. 4400 commits were contributed to the project over the past 7 years resulting in approximately 86 thousand lines of Java code excluding comments, empty lines and import statements. The release contains around 20 example machine learning tasks with 30 more experimental or larger-scale tasks contained in the project repository[4] (more than 150 if all variations are counted).

DL-Learner provides two command line scripts: One for the general execution of configuration files, which can solve standard machine learning tasks, and one tailored to knowledge base enrichment (as explained in Section 6.2). Moreover, it is possible to apply n-fold cross-validation by just adding one line to the config file, thus allowing for easy to use model validation for arbitrary learning tasks provided by DL-Learner. In addition, it is possible to set different fitness functions like F-measure or predictive accuracy, which allows learning algorithms to optimise for different criteria. The command line interface reads in configuration files, which are internally processed using the Spring Java Framework[5], and allow to configure almost arbitrary workflows as the code follows Java Beans conventions.

Another means to access DL-Learner, in particular for ontology engineering, is to use the Protégé plugin[6] (cf. Section 6.4). The plugin can suggest axioms to add to a knowledge base inside of the popular Protégé ontology editor.

Extending DL-Learner. DL-Learner is designed as a *framework* that is easily extensible allowing to add new custom components. As an example, to develop a new learning algorithm one has to implement the *class expression learning algorithm (CELA)* interface which means that certain methods to initialize, start and stop the algorithm have to be written, as well as methods to retrieve the best learned class expression. One simple example of the main actions performed by the start method could be to build up a certain set of complex concept expressions and exhaustively evaluate them with regards to an existing objective function (score) implemented in DL-Learner. This score of a concept expression depends on the actual problem to solve and is thus implemented in the applied *learning problem*. Using the framework has the advantage of reducing development time and it can be integrated into the command line interface almost effortlessly (just calling one method to register it).

In the same fashion, the framework can also be extended by new refinement operators which require to implement an initialization and refine method. The refine method takes a concept expression and dynamically derives a set of new expressions. Doing all the wiring and providing clear interfaces the DL-Learner framework allows extension developers to concentrate on their components and further eases an overall customization to solve dedicated learning tasks. By implementing the interfaces, algorithms can also readily be evaluated against all learning tasks provided in the framework.

6. Use Cases

6.1. Application to Life-Science Problems

In first experiments applying the concept learning approach to the life science domain, the DL-Learner was used to find descriptions of chemicals that might cause cancer. There have been several approaches by both – human and machines – to predict carcinogenicity. It has been shown in [16] that the performance of machine derived models for carcinogenicity can be equal to human experts. Classifying chemicals is a massive challenge, due to the high number and diversity of elements, structures, and tests involved in the problem. To provide experimental data, the U.S. National Toxicology Program (NTP) makes a set of more than 300 tested compounds available for training by Machine Learning tools. For each compound, the chemical structure, structural indicators, and the results of short-term assays were made available. In this *carcinogenesis prediction* task background knowledge about chemical compounds and existing results of already tested chemicals were used to derive complex descriptions of the cancer-causing characteristics. In a first step the original data[7], available in a Prolog-like syntax, was converted into an OWL ontology. To do this, we extended DL-Learner with a Prolog parser and wrote a mapping script to convert the carcinogenesis files into OWL. It is sometimes impossible and often not trivial to convert between both representations. For carcinogenesis such a mapping is possible, but required at least a superficial understanding of the domain. The mapping script we used and the resulting ontology are both freely available at the DL-Learner use-case description page.[8] During the transformation process almost no knowledge was lost or added. The resulting ontology contains 142 classes, 19 properties, 22 373 instances, and more than 74 000 facts, e.g. information about atoms like Gallium or Hydrogen, chemical structures like Halide or Methanol, and relates compounds to such structures or to its charge value. In the experiments, we run DL-Learner's OCEL algorithm with different settings and achieved a maximum cross validation accuracy of 67.4%, which outperformed other state-of-the-art approaches (cf. Table 1).

As an example, the following definition was obtained when using DL-Learner over the full training set (72% accuracy, for easier readability negation is moved further out if possible):

```
Compound and
not hasAtom some (Nitrogen-35 or Phosphorus-60 or
                  Phosphorus-61 or Titanium-134)
and (min 3 hasStructure (Halide and not Halide10)
     or (amesTestPositive = true and min 5 hasBond (
         not Bond-7)))
```

This can be phrased in natural language as follows:

[4] https://github.com/AKSW/DL-Learner

[5] http://projects.spring.io/spring-framework/

[6] http://dl-learner.org/community/protege-plugin/

[7] http://web2.comlab.ox.ac.uk/oucl/research/areas/machlearn/cancer.html

[8] http://dl-learner.org/community/carcinogenesis/

Table 1: Overview of the accuracy on carcinogenesis prediction achieved by different approaches using the same background knowledge

Approach	Accuracy (Std. Dev.)	Reference
DL-Learner (OCEL algorithm)	67.4% (±7.9%)	
Aleph with Ensembles	59.0% to 64.5%	[17]
Boosted Weak ILP	61.1%	[18]
Weak ILP	58.7%	[18]
Aleph Deterministic Top-Down 0.7	57.9% (±9.8%)	[19]
Aleph Randomized Rapid Restarts 0.9	57.6% (±6.4%)	[19]
Aleph Deterministic Top-Down 0.9	56.2% (±9.0%)	[19]
Aleph Randomized Rapid Restarts 0.7	54.8% (±9.0%)	[19]

"A chemical compound is carcinogenic iff it does not contain a Nitrogen-35, Phosphorus-60, Phosphorus-61, or Titanium-134 atom and it has at least three Halide – excluding Halide10 – structures or the ames test of the compound is positive and there are at least five atom bonds which are not of bond type 7."

The whole experimental setup as well as other experiments can be found in [9](cf. Sec. 7.2 and following).

In a further use case that is currently developed, the concept learning approach is applied to the prevalent gene and biomedical databases. Algorithms in DL-Learner are used to find descriptions for insulitis-related diseases based on background knowledge containing information about genes, gene functions and phenotypes. As a side effect, this use case is exploring scalability limits of all tools involved in the framework including reasoners. As a future direction the outcome of these investigations could be used for automated hypothesis generation or the extraction of new classifiers from biomedical knowledge bases. By exploiting large ontological background knowledge, the approaches may achieve more accurate and/or readable hypotheses for a number of frequent diseases.

6.2. Knowledge Base Enrichment

A standard use case for the learning algorithms contained in DL-Learner is *knowledge base enrichment*, i.e. the semi-automation of schemata creation and revision based on the available instance data. The combination of instance data and schemata allows improved querying, inference and consistency checking. 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 that the property `birthPlace` may be functional and has the domain `Person` as it is encoded via the following axioms in Manchester OWL syntax[9]:

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

Adding such axioms 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 (explicitly stated to be not the same) due to the functionality axiom. Specifically for the DBpedia knowledge base we observed an erroneous statement asserting that a person was born in Lacrosse, the game, instead of Lacrosse, 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. As an example in the above case it can be inferred that the birth place of a person is one of the places she stayed at. The main purpose of our research is to reduce the effort of creating and maintaining such schema information.

We have shown in [10] that the whole enrichment process can be described as illustrated in the lower part of Figure 4 and also be applied to large scale knowledge bases accessible via SPARQL. The general workflow proceeds in three steps:

1. In the optional first step, SPARQL queries are used to obtain existing information about the schema of 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.[10] Naturally, the schema only needs to be obtained once per knowledge base and can then be re-used by all algorithms and all entities in subsequent steps.
2. The second step consists of obtaining data via SPARQL which is relevant for learning the considered axiom. This results in a set of axiom candidates, configured via a threshold.
3. In the third step, the score of axiom candidates is computed and the results are returned.

In [11], patterns for frequent axioms are mined from more than one thousand ontologies and then used to learn on the DBpedia dataset. As one would expected, the most frequent axiom pattern was `A SubClassOf B`, but in the top 15 we found also patterns like

```
A SubClassOf p some (q some B)
A EquivalentTo B and p some C
A SubClassOf p value a
```

Those patterns have been applied to search for promising instantiations on DBpedia and resulted in axioms like

```
Song EquivalentTo MusicalWork and (artist some
                  Agent) and (writer some Artist)
```

or

```
Conifer SubClassOf order value Pinales
```

A manual evaluation with non-author experts judging the 2154 proposed axioms showed that 48.2% of these axioms were useful for extending the knowledge base. This shows promise, but also clearly indicates that a human expert is required in loop to ensure high quality.

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

[10]Note that the OWL reasoner only loads the schema of the knowledge base and, therefore, this option worked even in case with several hundred thousand classes in our experiments using the HermiT reasoner.

6.3. Knowledge Base Repair

While the number and size of datasets in the Semantic Web are increasing, there is also a risk for introducing modelling problems, e.g. because of data extraction errors or a *misunderstanding* and *misuse* of particular types of constructs in the ontology languages. Typical problems express themselves in *inconsistencies* or *unsatisfiable classes*. An inconsistency, in simple terms, is a logical contradiction in the knowledge base, which makes it impossible to derive any meaningful information by applying standard OWL reasoning techniques.[11] Unsatisfiable classes usually are a fundamental modelling error in that they cannot be used to characterize any individual.

Therefore, an application of DL-Learner that is directly accompanied with the enrichment use case (cf. Sec. 6.2) is ontology repair, i.e. detecting and fixing those problems. A core problem is usually that existing datasets are not expressive enough to apply ontology repair algorithms directly, which makes it interesting to apply knowledge base enrichment as a pre-processing step. While the enrichment process introduces uncertainty due to its machine learning nature, it frequently allows to detect a much wider range of problems. There are at least two tools which use DL-Learner as a library for ontology repair: ORE [20] and RDFUnit [21]. While ORE allows user feedback in a semi-automatic fashion, RDFUnit performs the process automatically but annotates axioms with confidence values from DL-Learner.

Another dimension of knowledge base repair can be achieved on the instance data level. Often, when DL-Learner algorithms suggest a possible axiom to add to the knowledge base, not all resources in the background knowledge fit that suggestion. Thus, one can directly report those instances (which are frequently corner-cases) to the user such that she can align the data, e.g. add missing data or correct existing data if possible.

6.4. Providing Suggestions in Ontology Editors

Together with the Protégé developers, we extended the Protégé 4/5 plugin mechanism to be able to seamlessly integrate the DL-Learner plugin as an additional method to create class expressions. This means that the knowledge engineer can use the algorithm exactly where it is needed without any additional configuration steps. The plugin has also become part of the official Protégé repository, i.e. it can be directly installed from within Protégé.

A screenshot of the plugin is shown in Figure 5. To use the plugin, the knowledge engineer is only required to press a button, which then starts the learning algorithm and during runtime periodically updates the list of suggested class expressions. For each suggestion, the plugin displays its accuracy.

When clicking on a suggestion, it is visualized by displaying two circles: One stands for the instances of the class to describe and another circle for the instances of the suggested class expression. Ideally, both circles overlap completely, but in practice this will often not be the case. Clicking on the plus symbol in each circle shows its list of individuals. Those individuals are also presented as points in the circles and moving the mouse over such a point shows information about the respective individual. Red points show potential problems detected by the plugin. Please note that we use closed world reasoning to detect those problems. If there is not only a potential problem, but adding the expression would render the ontology inconsistent, the suggestion is marked red and a warning message is displayed.

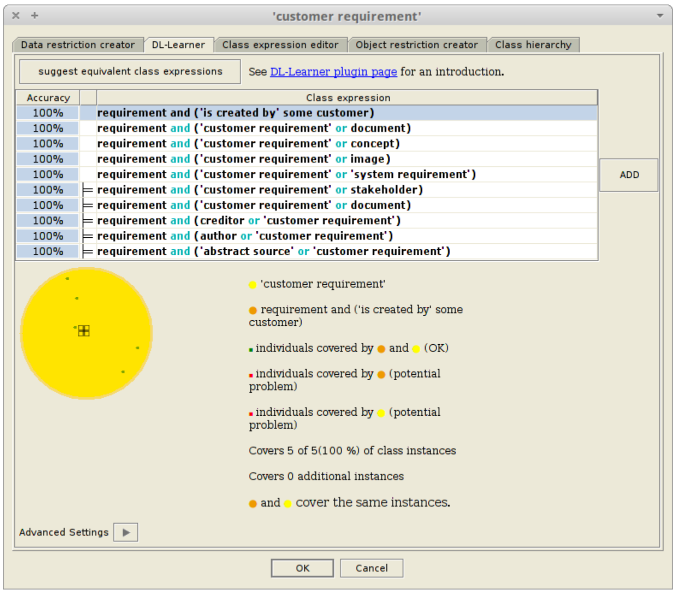


Figure 5: Screenshot of the DL-Learner Protégé plugin. Equivalent class expressions for the class *Customer Requirement* contained in the SWORE ontology v1.0[12]have been generated and the best one *Requirement and isCreatedBy some Customer* was selected showing no potential conflicts.

Accepting such a suggestion can still be a good choice, because the problem often lies elsewhere in the knowledge base, but was not obvious before, since the ontology was not sufficiently expressive for reasoners to detect it. This is illustrated in a screencast available on the plugin homepage[13], where the ontology becomes inconsistent after adding an axiom, and the real source of the problem is fixed afterwards. Being able to make such suggestions can be seen as a strength of the plugin.

The plugin allows the knowledge engineer to change expert settings. Those settings include the maximum suggestion search time, the number of results returned and settings related to the desired target language, e.g. the knowledge engineer can choose to stay within the OWL 2 EL profile or enable/disable certain class expression constructors. The learning algorithm is designed to be able to handle noisy data and the visualisation of the suggestions will reveal false class assignments so that they can be fixed afterwards.

We also plan to integrate the DL-Learner into WebProtégé[14] as it has full reasoning support. One use case related to this is that the DBpedia Association currently plans to adopt WebProtégé for maintaining the DBpedia ontology. DL-Learner in this

[11] There are indeed approaches that do some kind of approximate reasoning to infer useful information from an inconsistent ontology.

[12] SWORE Ontology: `http://ns.softwiki.de/req/`

[13] `http://dl-learner.org/community/protege-plugin/`

[14] `http://webprotege.stanford.edu/`

context would then serve as a component suggesting ontology extensions to all ontology maintainers for manual approval.

6.5. Knowledge Exploration

Typically, querying an RDF knowledge base via SPARQL queries is not considered an end user task as it requires familiarity with its syntax and the structure of the underlying knowledge base. For this reason, query interfaces are often tight to a specific knowledge base. More flexible techniques include facet based browsing, graphical query builders and Question Answering (QA) systems. We integrated the QTL algorithm (cf. Sec. 4) into the AutoSPARQL user interface [22]. It provides an alternative to the above interfaces with a different set of strengths and restrictions. AutoSPARQL uses active supervised machine learning to generate a SPARQL query based on positive examples, i.e. resources which should be in the result set of the SPARQL query, and negative examples, i.e. resources which should not be in the result set of the query. The user can either start with a question as in other QA systems or by directly searching for a relevant resource, e.g. "Rome". He then selects an appropriate result, which becomes the first positive example. After that, he is asked a series of questions on whether a resource, e.g. "London", should also be contained in the result set. These questions are answered by "yes" or "no". This feedback allows the supervised learning method to gradually learn which query the user is likely to be interested in. After each question is answered, QTL is invoked again, i.e. a new SPARQL query which returns all positive examples and none of the negative examples is returned. The user can always observe the result of the currently learned query and stop answering questions if the algorithm has correctly learned it. The system can also inform the user if there is no learnable query, which does not contradict with the selection of positive and negative examples. Overall, AutoSPARQL can generate more complex queries than facet-based browsers and most knowledge base specific applications, while still being easier to use than manual or visual SPARQL query builders and not much more difficult to use than facet-based browsers or standard QA systems.

6.6. Applications in other Domains

Besides the major use cases shown before, DL-Learner was also used by the community to solve interesting problems in other domains. We roughly describe two of them and refer the interested reader to the corresponding publications.

In [23], DL-Learner was used for *sentiment analysis*, a research area that analyses people's opinions or sentiments about e.g. products, services, individuals, events, etc. The idea is to pick some documents that reflect a positive opinion, annotate and translate those into OWL axioms by a novel vocabulary for texts, and finally apply the DL-Learner to learn class expressions that describe documents with a positive opinion.

[24] describes another application of the DL-Learner within the AORTA project[15], in which one of the primary goals is the optimization of the transportation workflow of patients as well as equipment in hospitals, which can drastically reduce costs. Given an ontology based on real-world datasets from hospitals describing all transport and related information over a longer timespan of several moths, first some semantic clustering is used to partition the data, and then on each partition, the DL-Learner is applied to learn rules. For instance, one explanation for why a transport is late might be

```
LateTransport EquivalentTo hasRoute some (
                           hasSegment some
                            {BusyElevator})
```

Those rules can then be taken into account when planning new transports.

[15]https://www.iminds.be/en/projects/aorta

7. Related Work

7.1. Inductive Learning

The goal of inductive learning approaches is to infer general principles from specific facts or instances usually considering some kind of background knowledge. Several systems using different techniques have been developed, for example Progol[16], Golem[17] and many others [2]. Those learning systems have been applied in several scenarios, e.g. learning drug structure activity rules, natural language processing, protein interaction [25, 26] or the detection of traffic problems. Overall, inductive reasoning is a rich and diverse research area with several applications, in particular those where rich structural knowledge is available.

Despite the relative success of inductive reasoning in some fields, there are also some open issues. One practical problem is to make results of ILP programs more readable and understandable for their users. This became evident for instance at the *Predictive Toxicology Challenge*[18] where different ILP programs could obtain promising results [27, 28, 29], however, they turned out to be hard to understand by domain experts. Another problem is scalability. Clearly, one issue in research and practice in ILP is its scalability to large datasets [30]. One challenge is the size of the hypothesis space, in particular for expressive languages.

7.2. Concept Learning in the Semantic Web

Whereas early approaches of applying machine learning techniques to Description Logics focused on the Probably Approximately Correct (PAC) [31] learnability of concept description languages later several supervised and unsupervised methods arose. One example of an unsupervised learning approach is the KLUSTER [32] algorithm which applies induction tasks to DL languages. Other than the DL-Learner framework, this algorithm is designed for general terminology induction and does not cover the tasks introduced in Section 2. Further research was done on operator-based techniques working on Description Logics [33, 34, 35, 36] which follow the idea of *generalization as search* [37]. These influenced the refinement operators available

[16]http://www.doc.ic.ac.uk/~shm/progol.html
[17]http://www.doc.ic.ac.uk/~shm/Software/golem/
[18]http://www.comlab.ox.ac.uk/activities/machinelearning/PTE/

in the DL-Learner framework as described in [9]. Besides the DL-Learner framework, the YinYang system [36] is another example of an implementation supporting concept learning on description logics. However, YinYang focused on learning $\mathcal{ALC}$ concepts and its algorithms tend to produce very long and hard-to-understand class expressions.

Another option for concept induction is offered by learning with logical decision trees [38] which were extended to support more expressive logical representations in clausal form [39]. With *terminological decision trees* [40] a model was introduced that allows the nodes of a logical decision tree to be terminological concept descriptions. The DL-Learner framework provides different implementations to learn with decision trees. Furthermore, DL-Foil [41], a new version of the Foil algorithm [42] was proposed which was adapted to learn concept descriptions supporting the OWL DL profile. The actual evaluation strategy of concept candidates in DL-Foil differs from the ideas presented in Section 2 since it follows the information gain approach of the Foil algorithm. Another algorithm that derives from Foil is Foil-$\mathcal{DL}$ [43] which adapts the refinement strategy and information gain calculation to learn fuzzy general class inclusion axioms from crisp $\mathcal{DL}$ knowledge bases.

8. Future Work

In future work, we target several advances related to horizontal scalability of the algorithms in DL-Learner. In particular, we will investigate algorithms for distributing the underlying data on several nodes of a cluster as well as distributing the hypothesis search via a master-slave mechanism on supercomputers. Within funded projects, DL-Learner will be evaluated in use cases related to IT monitoring, manufacturing workflows as well as compressor systems. For the application of DL-Learner in life science scenarios, we will research methods suitable for handling large schemata, i.e. those with many thousand classes and demanding requirements in terms of reasoning.

Acknowledgements. This work was supported by grants from the EU FP7 Programme for the project GeoKnow (GA no. 318159) as well as for the German Research Foundation project GOLD and the German BMWi project SAKE.

References

[1] T. Dietterich, P. Domingos, L. Getoor, S. Muggleton, P. Tadepalli, Structured machine learning: the next ten years, Machine Learning 73 (1) (2008) 3–23.

[2] S. Muggleton, L. De Raedt, D. Poole, I. Bratko, P. Flach, K. Inoue, A. Srinivasan, ILP turns 20, Machine Learning 86 (1) (2012) 3–23.

[3] J. Lehmann, DL-Learner: learning concepts in description logics, Journal of Machine Learning Research (JMLR) 10 (2009) 2639–2642.

[4] S.-H. Nienhuys-Cheng, R. de Wolf (Eds.), Foundations of Inductive Logic Programming, Vol. 1228 of LNCS, Springer, 1997.

[5] C. d'Amato, N. Fanizzi, F. Esposito, A note on the evaluation of inductive concept classification procedures, in: Proc. of the 5th Workshop on Semantic Web Applications and Perspectives (SWAP2008), Rome, Italy, 2008, Vol. 426, CEUR-WS.org, 2008.

[6] S. Hellmann, J. Lehmann, S. Auer, Learning of OWL class expressions on very large knowledge bases and its applications., in: Semantic Services, Interoperability and Web Applications: Emerging Concepts, IGI Global, 2011, pp. 104–130.

[7] J. Lehmann, S. Auer, L. Bühmann, S. Tramp, Class expression learning for ontology engineering, Journal of Web Semantics 9 (2011) 71 – 81.

[8] J. Lehmann, Hybrid learning of ontology classes, in: Proc. of the 5th Int. Conference on Machine Learning and Data Mining MLDM, Vol. 4571 of LNCS, Springer, 2007, pp. 883–898.

[9] J. Lehmann, P. Hitzler, Concept learning in description logics using refinement operators, Machine Learning journal 78 (1-2) (2010) 203–250.

[10] L. Bühmann, J. Lehmann, Universal OWL axiom enrichment for large knowledge bases, in: Proc. of EKAW 2012, 2012, pp. 57–71.

[11] L. Bühmann, J. Lehmann, Pattern based knowledge base enrichment, in: 12th International Semantic Web Conference, 21-25 October 2013, Sydney, Australia, 2013, pp. 33–48.

[12] J. Iglesias, J. Lehmann, Towards integrating fuzzy logic capabilities into an ontology-based inductive logic programming framework, in: Proc. of the 11th International Conference on Intelligent Systems Design and Applications (ISDA), 2011, pp. 1323–1328.

[13] J. Lehmann, C. Haase, Ideal downward refinement in the EL description logic, in: Inductive Logic Programming, 19th International Conference, ILP 2009, Leuven, Belgium, 2009, pp. 73–87.

[14] L. Bühmann, D. Fleischhacker, J. Lehmann, A. Melo, J. Völker, Inductive lexical learning of class expressions, in: Knowledge Engineering and Knowledge Management, Vol. 8876 of LNCS, Springer International Publishing, 2014, pp. 42–53.

[15] A. C. Tran, J. Dietrich, H. W. Guesgen, S. Marsland, An approach to parallel class expression learning, in: Proc. of the 6th International Conference on Rules on the Web: Research and Applications, RuleML'12, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 302–316.

[16] A. Srinivasan, R. D. King, D. W. Bristol, An assessment of submissions made to the predictive toxicology evaluation challenge, in: Proc. of the 16th International Joint Conference on Artifical Intelligence - Volume 1, IJCAI'99, 1999, pp. 270–275.

[17] I. Dutra, D. Page, V. S. Costa, J. Shavlik, An empirical evaluation of bagging in inductive logic programming, in: Proc. of the 12th International Conference on Inductive Logic Programming, Vol. 2583 of Lecture Notes in Artificial Intelligence, Springer, 2003, pp. 48–65.

[18] N. Jiang, S. Colton, Boosting descriptive ILP for predictive learning in bioinformatics, in: Proc. of the 15th International Conference on Inductive Logic Programming, Vol. 4455 of LNCS, Springer, 2006, pp. 275–289.

[19] F. Železný, A. Srinivasan, D. Page, Lattice-search runtime distributions may be heavy-tailed, in: Proc. of the 12th International Conference on Inductive Logic Programming, Vol. 2583 of Lecture Notes in Artificial Intelligence, Springer-Verlag, 2003, pp. 333–345.

[20] J. Lehmann, L. Bühmann, ORE - a tool for repairing and enriching knowledge bases, in: Proc. of the 9th International Semantic Web Conference (ISWC2010), LNCS, Springer, 2010, pp. 177–193.

[21] D. Kontokostas, P. Westphal, S. Auer, S. Hellmann, J. Lehmann, R. Cornelissen, A. Zaveri, Test-driven evaluation of linked data quality, in: Proc. of the 23rd International Conference on World Wide Web, WWW '14, 2014, pp. 747–758.

[22] J. Lehmann, L. Bühmann, Autosparql: Let users query your knowledge base, in: Proc. of ESWC 2011, 2011, pp. 63–79.

[23] A. Salguero, M. Espinilla, Sentiment Analysis and Ontology Engineering: An Environment of Computational Intelligence, 2016, Ch. Description Logic Class Expression Learning Applied to Sentiment Analysis, pp. 93–111.

[24] P. Bonte, F. Ongenae, F. D. Turck, Learning semantic rules for intelligent transport scheduling in hospitals, in: Proc. of the 5th Workshop on Data Mining and Knowledge Discovery meets Linked Open Data, 2016.

[25] J. Chen, L. Kelley, S. Muggleton, M. Sternberg, Protein fold discovery using stochastic logic programs, in: Probabilistic inductive logic programming, Springer, 2008, pp. 244–262.

[26] J. C. Santos, H. Nassif, D. Page, S. H. Muggleton, M. J. Sternberg, Automated identification of protein-ligand interaction features using inductive logic programming: a hexose binding case study, BMC bioinformatics 13 (1) (2012) 162.

[27] R. Benigni, A. Giuliani, Putting the predictive toxicology challenge into perspective: Reflections on the results, Bioinformatics 19 (10) (2003) 1194–1200.

[28] C. Helma, R. D. King, S. Kramer, A. Srinivasan, The predictive toxicology challenge 2000-2001, Bioinformatics 17 (1) (2001) 107–108.

[29] H. Toivonen, A. Srinivasan, R. D. King, S. Kramer, C. Helma, Statistical

evaluation of the predictive toxicology challenge 2000-2001, Bioinformatics 19 (10) (2003) 1183–1193.

[30] H. Watanabe, S. Muggleton, Can ILP be applied to large dataset?, in: Inductive Logic Programming, 19th International Conference, ILP 2009, Leuven, Belgium, 2009, pp. 249–256.

[31] A. Blumer, A. Ehrenfeucht, D. Haussler, M. K. Warmuth, Learnability and the vapnik-chervonenkis dimension, Journal of the ACM (JACM) 36 (4) (1989) 929–965.

[32] J. U. Kietz, K. Morik, A polynomial approach to the constructive induction of structural knowledge, Machine Learning 14 (2) (1994) 193–218.

[33] N. Fanizzi, F. Esposito, S. Ferilli, G. Semeraro, A methodology for the induction of ontological knowledge from semantic annotations, in: Proc. of the Conf. of the Italian Association for Artificial Intelligence, Vol. 2829 of LNAI/LNCS, Springer, 2003, pp. 65–77.

[34] F. Esposito, N. Fanizzi, L. Iannone, I. Palmisano, G. Semeraro, Knowledge-intensive induction of terminologies from metadata, in: The Semantic Web – ISWC 2004: Third International Semantic Web Conference. Proceedings, Springer, 2004, pp. 441–455.

[35] N. Fanizzi, S. Ferilli, L. Iannone, I. Palmisano, G. Semeraro, Downward refinement in the $\mathcal{ALN}$ description logic, in: Proc. of the 4th International Conference on Hybrid Intelligent Systems, HIS2004, IEEE Computer Society, 2005, pp. 68–73.

[36] L. Iannone, I. Palmisano, N. Fanizzi, An algorithm based on counterfactuals for concept learning in the semantic web, Applied Intelligence 26 (2) (2007) 139–159.

[37] T. M. Mitchell, Generalization as search, Artificial Intelligence 18 (2) (1982) 203–226.

[38] J. R. Quinlan, Induction of decision trees, Machine Learning 1 (1986) 81–106.

[39] H. Blockeel, L. D. Raedt, Top-down induction of first-order logical decision trees, Artificial Intelligence 101 (1-2) (1998) 285–297.

[40] N. Fanizzi, C. d'Amato, F. Esposito, Induction of concepts in web ontologies through terminological decision trees, in: Proc. of ECML PKDD 2010, Part I, Vol. 6321 of LNCS/LNAI, Springer, 2010, pp. 442–457.

[41] N. Fanizzi, C. d'Amato, F. Esposito, DL-FOIL: Concept learning in description logics, in: Proc. of the 18th International Conference on Inductive Logic Programming, ILP2008, Vol. 5194 of LNAI, Springer, 2008, pp. 107–121.

[42] J. R. Quinlan, Learning logical definitions from relations, Machine Learning 5 (1990) 239–266.

[43] F. A. Lisi, U. Straccia, Learning in description logics with fuzzy concrete domains, Fundamenta Informaticae 140 (3-4) (2015) 373–391.