DL-Learner – Structured Machine Learning on Semantic Web Data

Lorenz Bühmann
SDA Research Group
Institute for Applied Informatics, University of Leipzig
Leipzig, Germany
buehmann@informatik.uni-leipzig.de

Jens Lehmann
Enterprise Information Systems, Fraunhofer IAIS
Institute of Computer Science, University of Bonn
Sankt Augustin, Germany and Bonn, Germany
jens.lehmann@cs.uni-bonn.de

Patrick Westphal
SDA Research Group
Institute for Applied Informatics, University of Leipzig
Leipzig, Germany
patrick.westphal@informatik.uni-leipzig.de

Simon Bin
SDA Research Group
Institute for Applied Informatics, University of Leipzig
Leipzig, Germany
sbin@informatik.uni-leipzig.de

ABSTRACT

The following paper is an extended summary of the journal paper [5]. In this system paper, we describe the DL-Learner framework. It is beneficial in various data and schema analytic tasks with applications in different standard machine learning scenarios, e.g. 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.

CCS CONCEPTS

• **Information systems → Web Ontology Language (OWL)**; *Resource Description Framework (RDF)*; • **Computing methodologies → Structured outputs**; **Inductive logic learning**; *Supervised learning by classification*; *Statistical relational learning*; *Rule learning*; *Instance-based learning*;

KEYWORDS

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

ACM Reference Format:
Lorenz Bühmann, Jens Lehmann, Patrick Westphal, and Simon Bin. 2018. DL-Learner – Structured Machine Learning on Semantic Web Data. In *WWW '18 Companion: The 2018 Web Conference Companion, April 23–27, 2018, Lyon, France,* Claudia d'Amato, Francesco Marcelloni, and Rudi Studer (Eds.). ACM, New York, NY, USA, 5 pages. https://doi.org/https://doi.org/10.1145/3184558.3186235

WWW '18 Companion, April 23–27, 2018, Lyon, France

ACM ISBN 978-1-4503-5640-4.
https://doi.org/https://doi.org/10.1145/3184558.3186235

1 INTRODUCTION

Over the past two decades, data and knowledge have become more important in our society. A major challenge that research faces today is to analyse this growing amount of information to obtain insights into the underlying problems, e.g. in [7] and [18]. 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[1] 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.

A previous system paper on DL-Learner appeared in 2009 in the Journal of Machine Learning Research [12]. Compared to that system description, the 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. Moreover, algorithms are continuously extended with options based on received feature requests. In the same manner, new APIs and reasoners are continuously upgraded.

• 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.

[1] http://dl-learner.org, https://github.com/SmartDataAnalytics/DL-Learner

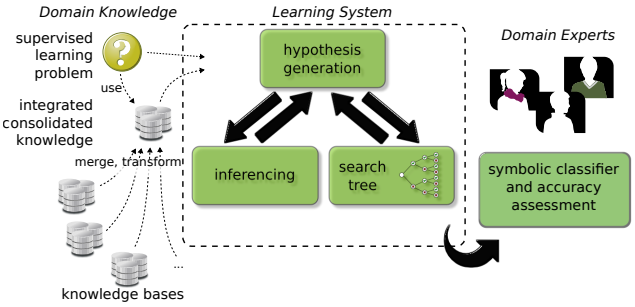

Figure 1: General learning workflow.

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. Use cases of DL-Learner in different problem areas are covered in Section 5. Notes regarding implementation and extension, as well as related and future work are discussed in the journal article [5].

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*. Learning problems which are similar to the one we will analyse, have been investigated in *Inductive Logic Programming* [19].

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.

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

Standard Supervised Learning Let the name of the back-ground 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. 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).

DL-Learner implements standard binary measures, e.g. predictive accuracy, F-measure, and all ternary measures described in [6]. 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.

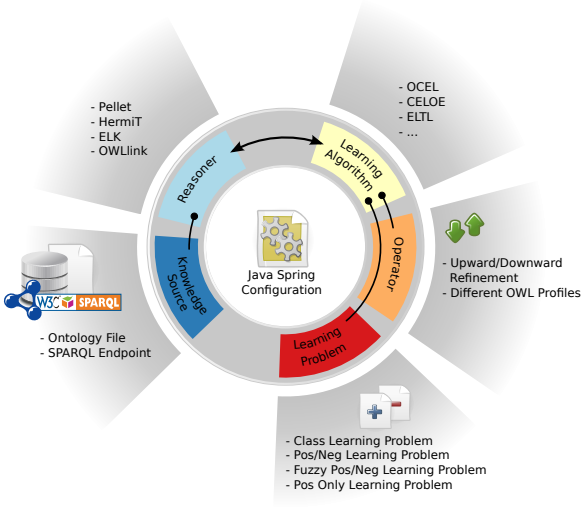

Figure 2: Overview of core DL-Learner components.

3 OVERVIEW OF THE FRAMEWORK

The DL-Learner framework provides means to flexibly build concept learning algorithms. Several (Java) interfaces, adaptors 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 [8] 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 approximate 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

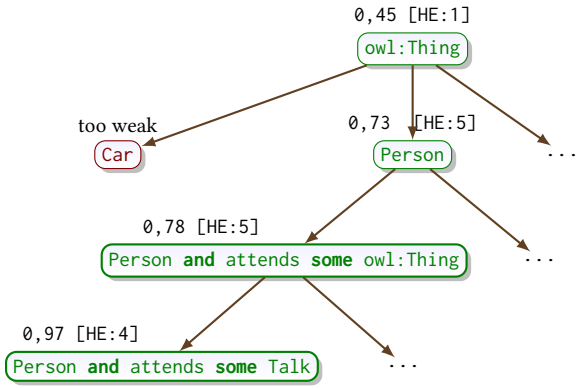

Figure 3: Search tree used in OCEL and CELOE algorithm.

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 [13]. 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.

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 [11]. This was extended to very expressive schemata [17] and learning problems with a lot of instance data [8]. Later, we extended the theoretical and algorithmic foundations for a) learning complex definitions in ontologies [13], b) generic schema enrichment [3, 4], c) fuzzy description logics [9], d) the light-weight $\mathcal{EL}$-description logic [16] and e) combinations of natural language processing and concept learning [2]. 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*[2] *search space* $(\Sigma, \preceq)$

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

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.

A summary of the *CELOE* (Class Expression Learning for Ontology Engineering) [13], *ELTL* (EL Tree Learner) [16] and *ISLE* (Inductive Statistical Learning of Expressions) [2], as well as *OWL Schema Learning Algorithms* [3] and *axiom pattern enrichment* [4], *QTL* (Query Tree Learning), *PARCEL* [22], *Fuzzy DLL* [9] and genetic algorithms [11] can be found in the journal version [5].

5 USE CASE: 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[3]:

```
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

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

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

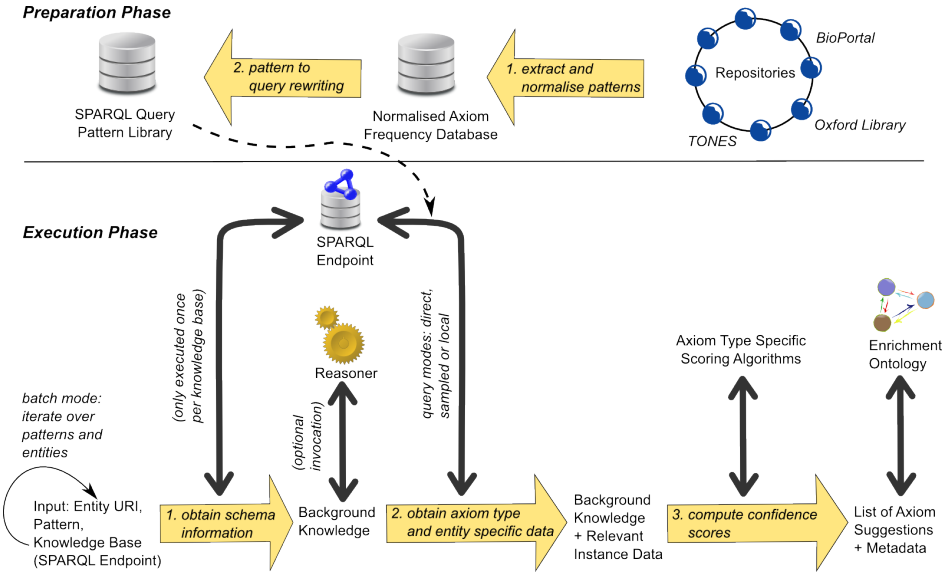

Figure 4: The general workflow of knowledge base enrichment and its pattern based extension: Frequent axiom patterns in various ontology portals are detected and converted into SPARQL query patterns (upper part). Those are then applied to other data-sets to enrich them with further axioms (lower part).

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 [3] 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.[4] 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 [4], patterns for frequent axioms are mined from more than one thousand ontologies and then used to learn on the DBpedia data-set. 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.

Many more use cases such as Life-Science Problems [21] like *carcinogenesis prediction*, *Ontology Repair* (ORE [14], RDFUnit [10]), Suggestions for Ontology Editors[5], *Knowledge Exploration* [15], *sentiment analysis* [20], or hospital workflow management [1] are summarised in the journal version [5].

ACKNOWLEDGMENTS

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 Ministry for Economic Affairs and Energy project SAKE (GA no. 01MD15006E) and the European Union's Horizon 2020 research and innovation programme for the project SLIPO (GA no. 731581).

[4]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.

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

REFERENCES

[1] Pieter Bonte, Femke Ongenae, and Filip De Turck. 2016. 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*.

[2] Lorenz Bühmann, Daniel Fleischhacker, Jens Lehmann, Andre Melo, and Johanna Völker. 2014. Inductive Lexical Learning of Class Expressions. In *Knowledge Engineering and Knowledge Management (LNCS)*, Vol. 8876. Springer International Publishing, 42–53.

[3] Lorenz Bühmann and Jens Lehmann. 2012. Universal OWL Axiom Enrichment for Large Knowledge Bases. In *Proc. of EKAW 2012*. 57–71.

[4] Lorenz Bühmann and Jens Lehmann. 2013. Pattern Based Knowledge Base Enrichment. In *12th International Semantic Web Conference, 21–25 October 2013, Sydney, Australia*. 33–48.

[5] Lorenz Bühmann, Jens Lehmann, and Patrick Westphal. 2016. DL-Learner – A framework for inductive learning on the Semantic Web. *Web Semantics: Science, Services and Agents on the World Wide Web* 39 (2016), 15–24.

[6] Claudia d'Amato, Nicola Fanizzi, and Floriana Esposito. 2008. 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.

[7] Thomas Dietterich, Pedro Domingos, Lise Getoor, Stephen Muggleton, and Prasad Tadepalli. 2008. Structured machine learning: the next ten years. *Machine Learning* 73, 1 (October 2008), 3–23.

[8] Sebastian Hellmann, Jens Lehmann, and Sören Auer. 2011. Learning of OWL Class Expressions on Very Large Knowledge Bases and its Applications. In *Semantic Services, Interoperability and Web Applications: Emerging Concepts*. IGI Global, 104–130.

[9] Josué Iglesias and Jens Lehmann. 2011. 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)*. 1323–1328.

[10] Dimitris Kontokostas, Patrick Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, and Amrapali Zaveri. 2014. Test-driven Evaluation of Linked Data Quality. In *Proc. of the 23rd International Conference on World Wide Web (WWW'14)*. 747–758.

[11] Jens Lehmann. 2007. Hybrid Learning of Ontology Classes. In *Proc. of the 5th Int. Conference on Machine Learning and Data Mining MLDM (LNCS)*, Vol. 4571. Springer, 883–898.

[12] Jens Lehmann. 2009. DL-Learner: Learning Concepts in Description Logics. *Journal of Machine Learning Research (JMLR)* 10 (2009), 2639–2642.

[13] Jens Lehmann, Sören Auer, Lorenz Bühmann, and Sebastian Tramp. 2011. Class expression learning for ontology engineering. *Journal of Web Semantics* 9 (2011), 71–81.

[14] Jens Lehmann and Lorenz Bühmann. 2010. ORE – A Tool for Repairing and Enriching Knowledge Bases. In *Proc. of the 9th International Semantic Web Conference (ISWC2010) (LNCS)*. Springer, 177–193.

[15] Jens Lehmann and Lorenz Bühmann. 2011. AutoSPARQL: Let Users Query Your Knowledge Base. In *Proc. of ESWC 2011*. 63–79.

[16] Jens Lehmann and Christoph Haase. 2009. Ideal Downward Refinement in the EL Description Logic. In *Inductive Logic Programming, 19th International Conference, ILP 2009, Leuven, Belgium*. 73–87.

[17] Jens Lehmann and Pascal Hitzler. 2010. Concept Learning in Description Logics Using Refinement Operators. *Machine Learning journal* 78, 1–2 (2010), 203–250.

[18] Stephen Muggleton, Luc De Raedt, David Poole, Ivan Bratko, Peter Flach, Katsumi Inoue, and Ashwin Srinivasan. 2012. ILP turns 20. *Machine Learning* 86, 1 (2012), 3–23.

[19] Shan-Hwei Nienhuys-Cheng and Ronald de Wolf (Eds.). 1997. *Foundations of Inductive Logic Programming*. LNCS, Vol. 1228. Springer.

[20] Alberto Salguero and Macarena Espinilla. 2016. *Sentiment Analysis and Ontology Engineering: An Environment of Computational Intelligence*. Chapter Description Logic Class Expression Learning Applied to Sentiment Analysis, 93–111.

[21] A. Srinivasan, R. D. King, and D. W. Bristol. 1999. 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)*. 270–275.

[22] An C. Tran, Jens Dietrich, Hans W. Guesgen, and Stephen Marsland. 2012. 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, 302–316.