SML-Bench – A Benchmarking Framework for Structured Machine Learning

Patrick Westphal[a] Lorenz Bühmann[a] Simon Bin[a] Hajira Jabeen[b] Jens Lehmann[b,c]
[a] *Computer Science Institute, University of Leipzig, {pwestphal,buehmann,sbin}@informatik.uni-leipzig.de*
[b] *Computer Science Institute, University of Bonn, {jabeen, jens.lehmann}@cs.uni-bonn.de*
[c] *Fraunhofer Institute for Intelligent Analysis and Information Systems (IAIS), jens.lehmann@iais.fraunhofer.de*

Abstract. The availability of structured data has increased significantly over the past decade and several approaches to learn from structured data have been proposed. These logic-based, inductive learning methods are often conceptually similar, which would allow a comparison among them even if they stem from different research communities. However, so far no efforts were made to define an environment for running learning tasks on a variety of tools, covering multiple knowledge representation languages. With SML-Bench, we propose a benchmarking framework to run inductive learning tools from the ILP and semantic web communities on a selection of learning problems. In this paper, we present the foundations of SML-Bench, discuss the systematic selection of benchmarking datasets and learning problems, and showcase an actual benchmark run on the currently supported tools.

Keywords: benchmark, structured machine learning

1. Introduction

With the growth of the number and size of data sources over the last years, there is an increasing demand for algorithms and tools to perform accurate analysis of these datasets. History in computer science has shown that the main driver to scientific advances, and in fact a core element of the scientific method as a whole, is the provision of benchmarks to make progress measurable. A famous example from database benchmarking (specifically TPC-A[1]) is considered to have been the motor that improved transaction performance of relational databases by an order of magnitude on equal hardware in the 90s. Other more recent benchmarking areas related to semantic technologies have been ‘question answering benchmarks’ (QALD [19]), ontology matching (OAEI[2]), as well as graph and triple store query performance (LDBC[3]). All of those have led to significant performance improvements.

One area, which is not extensively covered by benchmarks yet is symbolic supervised machine learning from structured data. In this task, background knowledge is modelled using RDF, OWL, Prolog, or other knowledge representation languages. Within this background knowledge, entities are selected as positive and negative examples (supervised learning). Based on those examples, logical formulas e.g. Horn rules or OWL class expressions are learned, using some algorithms, which separate positive and negative examples. These formulas or rules are later used to predict further unseen entities. Hence, this supervised machine learning discipline considers classification methods that generate *symbolic* (as opposed to *numeric*) classifiers which describe distinctive structures of the underlying knowledge base w.r.t. the given positive (and

[1]http://www.tpc.org/tpca/default.asp
[2]http://oaei.ontologymatching.org/
[3]http://ldbc.eu

negative) examples. For instance, given a dataset describing chemical compounds in which positive examples are compounds known to cause cancer and negative examples are compounds which do not cause cancer, the algorithms would induce formulas describing what causes cancer. An example result of an OWL class expression describing carcinogenic compounds would be[4]

$$
\begin{aligned}
&(\mathsf{Compound} \sqcap \neg\exists \mathsf{hasAtom}.(\\
&\qquad \mathsf{Nitrogen\text{-}35} \sqcup \mathsf{Phosphorus\text{-}60} \\
&\qquad \sqcup \mathsf{Phosphorus\text{-}61} \sqcup \mathsf{Titanium\text{-}134}) \\
&\quad \sqcap (\geq 3\ \mathsf{hasStructure}.(\mathsf{Halide} \sqcap \neg \mathsf{Halide10}) \\
&\qquad \sqcup (\mathsf{amesTestPositive} = \mathit{true} \\
&\qquad\quad \sqcap \geq 5\ \mathsf{hasBond}.(\neg \mathsf{Bond\text{-}7}))))
\end{aligned}
$$

which can be verbalised as ‘a chemical compound that does not contain a Nitrogen-35, Phosphorus-60, Phosphorus-61, or Titanium-134 atom, and which 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’. A very simple strategy for an algorithm to derive such expressions could be to group the positive examples into subsets of ‘similar’ examples that can be generalized by a preferably specific classifier, and in a later step combine those ‘sub-classifiers’ to the overall classifier for the learning problem at hand. However, there are many different approaches and strategies as mentioned later. Two major advantages of those methods are that 1.) they can work with complex background knowledge including inference and 2.) the result can be interpreted and understood by humans.

While a large body of research work has been devoted to this area, the evaluation scenarios are scattered and no generally accepted reference benchmarking platform exists. There are at least two major reasons for this: 1.) The first problem is that the use of different knowledge representation languages makes results very difficult to compare. For instance, logic programs are incomparable in terms of expressivity to the description logics underlying OWL. In practice, the knowledge modelling is different to such an extent, that automatic conversion methods do not output satisfactory results. 2.) The second reason is that the effort required to model the background knowledge using semantic technologies can be considered significantly high. Besides the necessary domain expertise, a basic understanding of ontology engineering principles and knowledge of basic vocabularies and existing complementary background knowledge bases is required. Applying this to compile knowledge bases for many learning problems from different domains and setting up a repository is a major effort. Overall, this has led to benchmarks being scattered across different publications and scientific communities.

To overcome this problem, we have performed a systematic scientific literature analysis in order to collect relevant benchmarks. Those were then translated into different knowledge representation languages if needed. For the execution of benchmarks, a framework has been implemented, which allows the execution of different learning systems over a given set of learning tasks and measure performance metrics. Moreover, wrappers for the systems Progol[5], Golem[6], Aleph[7], FuncLog[8], TopLog[8], ProGolem[8], TreeLiker[9] and DL-Learner[10] have been written to include them in the framework. Overall, our main contributions are:

- A systemic survey of articles published in the last 10 years in relevant scientific conferences and journals collecting benchmarks for structured machine learning.
- The preparation of nine learning tasks, which constitute our current benchmark suite, including translations of the background knowledge in OWL and different logic programming dialects.
- The creation of a framework (called SML-Bench), which allows comparison of systems which differ in the knowledge representation languages they support, and the programming languages they are written in.

[4]See `http://dl-learner.org/community/carcinogenesis/` for a deeper discussion on that particular topic.

[5]`http://www.doc.ic.ac.uk/~shm/progol.html`
[6]`http://www.doc.ic.ac.uk/~shm/golem.html`
[7]`http://www.cs.ox.ac.uk/activities/machinelearning/Aleph/`
[8]`http://www.doc.ic.ac.uk/~jcs06/GILPS/`
[9]`http://ida.felk.cvut.cz/treeliker/TreeLiker.html`
[10]`http://dl-learner.org`

– The creation of wrappers for eight learning systems for their inclusion in SML-Bench.

The paper is structured as follows: In Section 2 we give an overview of related work, followed by sections introducing the challenges (Section 3) and benchmarking strategies (Section 4) of structured machine learning. We further describe our dataset review process in Section 5. In Section 6 we introduce our benchmark framework and describe our evaluation setup and results in Section 7. After a discussion in Section 8 we give an outlook and conclude our paper in Section 9.

2. Related Work

Machine learning is a vast field with a variety of application domains and learning problems. Existing benchmark initiatives can be categorised based on data sizes, learning problems, and data types (associated with a particular set of algorithms). However not all of those categories have to apply. For instance, popular dataset repositories like UCI [5], StatLog [11] and StatLib [27] – though they provide datasets with evaluation results of individual algorithms – have the major aim of providing data, and do not provide comprehensive comparisons of different algorithms.

A noticeable effort for benchmarking, both datasets and algorithms, can be seen in LIBSVM[11] [6]. The authors have collected datasets from various repositories and preprocessed them for compatibility with LIBSVM. Most of the datasets in the above mentioned repositories are in tabular or CSV format and the underlying structure of the data is often flat and simple, rendering them beyond the scope of SML-Bench.

RLBench[12], on the other hand, emphasises a particular task i.e. reinforcement learning. It aims at implementing reinforcement learning algorithms in a uniform manner so that new algorithms can be easily tested on a set of different problems. The learning problems in this benchmark are synthetically generated and contain reinforcement learning specific parameters like state, rewards, or actions etc.

A benchmark focusing on neural networks is *Shirley's Next-Generation Benchmark Suite*[13]. It includes numerous types of neural networks that perform different tasks on the provided data. The benchmark suite contains several datasets collected from different internet resources together with results from simulation of different machine learning algorithms. This covers bitmap or numerical data tailored for certain neural network techniques which is not the focus of our work.

Recently, benchmarking and data collection attempts have started to cover big data and scalability tests. Bench-ML[14] provides a minimal benchmark of learning tools for preprocessing, visualisation and machine learning algorithms for commonly used open source implementations e.g. in R, Python, H2O and Spark-MLlib. This benchmark can be used for testing scalability, speed and accuracy of the above mentioned machine learning tools. However, it uses only one dataset from an airline for the evaluation that is tabular and lacks semantic structure. A similar effort[15] assesses Redshift, Hive, Shark, Impala and Stinger, with the goal of providing understandable and reproducible results. The data includes unstructured HTML documents and tables containing summary information. Fox et al. [8] have provided a benchmark for HPC applications by providing multiple *Ogres* or facets for understanding them. The work of Ming et al. [20] focuses on a generator that creates structured, semi-structured and unstructured data such as text, graph, or tables for numerous big data related tasks reflecting properties of real world datasets. In a Spark specific benchmarking suite [18], a variety of different algorithms related to machine learning (logistic regression, SVM, matrix factorisation), graph computation (PageRank, SVD++, TriangleCount), SQL queries and streaming applications are included. The evaluation is done using synthetic datasets with a variety of workloads. Some other big data related benchmarks are [9,10,4] and [3]. The datasets provided by all of the above mentioned benchmarking efforts are not sufficiently structured and cannot be used directly.

One of the noticeable benchmarks in the semantic web community for query performance is proposed by the *Linked Data Benchmark Coun-*

[11]http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/
[12]http://hunch.net/~rlbench/
[13]http://lava.cs.virginia.edu/shirley_benchmark/
[14]https://github.com/szilard/benchm-ml
[15]https://amplab.cs.berkeley.edu/benchmark/

cil (LDBC)[16] [2]. It comprises two sets of benchmarks covering semantic publishing and social networks. RdfStoreBenchmarking[17] is a repository that provides an exclusive collection of references to RDF benchmarks, benchmarking results and papers about RDF benchmarking. The main focus of these benchmarking efforts is to collect different types of RDF data or to provide meaningful schema information. It covers tasks like evaluating the query performance of different semantic web repositories that provide a SPARQL endpoint, or use of graphs with different properties (e.g. regarding their connectedness), or linked data translation and integration, performance evaluation of federated query engines, linked data quality assessment or data fusion systems. It can be noted that, most of the systems provided at the above repository are concerned with basic triple storage and retrieval performance related tasks. On the other hand, we are interested in (structured) data sets that are particularly suitable for supervised machine learning tasks. We require that the data allows to derive a classification problem as described in Section 1. In Table 1, we summarize the benchmarking efforts described in this section.

As we focus on symbolic machine learning from examples and semantically structured background knowledge, the benchmarking projects introduced here are not suitable. None of them deals with semantically structured data and/or algorithms, that explicitly present a supervised structured machine learning problem, making our benchmarking effort considerably different and unique.

3. Challenges of Structured Machine Learning

In this section, we describe some of the challenges of machine learning on structured data grouped by category:

Background Language Expressivity A powerful property of machine learning algorithms operating on structured data is their ability to reason about the background knowledge. Generally, more expressive languages for the background knowledge, e.g. an expressive description logic such as $\mathcal{SROIQ}$, have a higher (worst case) time complexity than a lightweight language. Structured machine learning algorithms usually either include a reasoner as part of their architecture or integrate reasoning capabilities into the core of their algorithms (sometimes without completeness and correctness guarantees for results).

Size Larger datasets affect the efficiency of structured machine learning tools. There are three main aspects of size: (1) The size of the schema, (2) the size of the instance data and (3) the number of examples. The first aspect, schema size, affects mainly the *hypothesis space* as the learned concepts are constructed from the available schema. The schema size is the number of predicates in the background knowledge (where OWL classes are unary and OWL properties are treated as binary predicates). In the absence of a particular language bias, e.g. restrictions on the length of learned concepts, nested concepts etc., the size of the hypothesis space can become extremely large for big schemata. The second and third aspect, i.e. the instance data size and number of examples, affect mainly the time for *hypothesis checking*, i.e. validating whether a concept fits the given examples well.

Target Concept Language Given the same background knowledge, the performance of a structured machine learning algorithm will still heavily depend on the target concept language which is not necessarily the same as the background knowledge language. For instance, some algorithms can learn arbitrarily nested predicate structures (e.g. "parents having studied in Germany" could be represented via nesting of the predicates `parent` and `studied`). Moreover, in particular for description logics, available concept constructors, such as existential and universal quantification or qualified cardinality restrictions, can be included or excluded. Including them increases the search space, but potentially also allows to find better solutions.

Within our evaluation, we will discuss how the tools perform on the selected datasets with respect to the above challenges (often also referred to as *choke points* in the benchmarking literature).

4. Benchmarking Structured Machine Learning Algorithms

As introduced in Section 1 the general task of an inductive learning algorithm is to learn a classifier

[16]http://ldbcouncil.org/industry/organization/origins
[17]https://www.w3.org/wiki/RdfStoreBenchmarking

Table 1
Comparison of different benchmarking efforts

Benchmark	Data type	Algorithms	Learning Problems/Benchmarking Tasks
SML-Bench	Structured data	Multiple SML algorithms	Structured supervised ML
UCI, StatLog, StatLib	Tabular data	No algorithms	Variety of learning problems
LIBSVM	Tabular data	SVM and others	Machine learning
Shirleys NN	Tabular data	Neural networks	Classification
RLBench	Tabular data	Reinforcement learning	Reinforcement learning
Bench-ML, Amplab	Large datasets	Scalable processing	Machine learning, query performance
Ogres	Large datasets	Scalable processing	HPC applications
LDBC	Linked Data	Benchmarks	Query performance, quality assessment, etc

discriminating two classes (generally referred to as *positive* and *negative*) based on distinguished examples and some background knowledge. In case of the symbolic learning approaches considered here those classifiers are, for example, Horn rules or description logic class expressions. For benchmarking, the respective implementations of the considered algorithms are taken as *'black boxes'*, i.e. no evaluations on algorithmic or implementation details are performed. To judge the quality of a binary classifier there are several measures that are all derived from the *confusion matrix* [26] which provides the numbers of examples correctly classified as positive, incorrectly classified as positive, correctly classified as negative and incorrectly classified as negative. Derived quality metrics are, e.g. accuracy, precision, recall, F-score, specificity, and AUC [26]. In Section 7 we focus on average accuracy and average F_1-score for brevity.

Considering more general performance indicators of software systems [28], measures that might apply in the structured machine learning context are memory usage and the overall runtime. Taking the overall runtime into consideration does not allow a complete comparison across all learning systems since some of them implement anytime algorithms (which terminate after a pre-set execution time). Thus, even though the overall runtime values are provided with the benchmark results, in Section 7 we only report when systems could not finished within the maximum execution time configured in the overall benchmark settings.

In terms of memory usage, measuring the space complexity of a certain algorithm would be a valuable metric for comparison. However, since the respective algorithms are implemented using different programming languages and execution environments, determining the actual memory usage is not an easy task. First of all, to run a set of such heterogeneous implementations we cannot invoke them *inside* one single benchmarking system but have to execute the implementations as own processes in their native environments, like, for example, a Prolog interpreter, a Java Virtual Machine etc. This means that specialized memory analysis tools like Valgrind[18] would have to be interposed. However, those can only measure the RAM usage of the respective execution environments which have their own specifics, like, for example, an interpreter's object model and memory layout, the libraries loaded etc. Furthermore, taking into consideration that different knowledge representation languages come with their own representational overhead (compare, for example, relatively short atom names in Prolog with the full IRIs used in OWL) makes the reliable assessment of the memory usage a non-trivial task. Due to these practical difficulties we currently do not report the memory usage but just note in the benchmark results when a learning system ran out of memory.

Assessing the scalability of a structured machine learning algorithm along the dimensions reported in Section 3 would require a learning problem generator which can create synthetic training data which may differ in terms of its knowledge representation language expressivity, number of schema axioms/rules, number of assertions/facts, number of examples, and the expressivity of the target concept language. We consider the problem of designing such a learning problem generator, which provides freely scalable, sound, meaningful and non-biased learning problems a non-trivial task which was not investigated in depth, yet, and thus requires further research. Hence, we leave this as future work.

[18] http://valgrind.org/

Table 2
Conferences and journals with number of surveyed papers (#P) and candidate datasets (#C)

Source	Year	#P	#C	Source	Year	#P	#C
ECML	2006	39	0	JMLR	2006	100	0
	2007	28	2		2007	91	0
	2008	25	0		2008	104	1
	2009	105	0		2009	100	0
	2010	120	0		2010	118	1
	2011	121	2		2011	105	0
	2012	128	3		2012	119	0
	2013	158	10		2013	118	2
	2014	147	8		2014	118	1
	2015	139	8		2015	118	1
ILP	2006	27	1	KDD	2006	126	3
	2007	22	0		2007	116	0
	2008	40	14		2008	119	0
	2009	40	8		2009	145	0
	2010	31	5		2010	136	0
	2011	66	7		2011	178	0
	2012	35	12		2012	210	0
	2013	31	14		2013	197	0
	2014	40	10		2014	218	5
	2015	36	13		2015	253	6
ICML	2006	140	0	MLJ	2006	55	0
	2007	150	0		2007	46	0
	2008	158	0		2008	55	0
	2009	159	0		2009	52	13
	2010	159	0		2010	61	2
	2011	160	0		2011	58	0
	2012	247	0		2012	55	2
	2013	283	0		2013	61	0
	2014	310	0		2014	63	0
	2015	270	0		2015	80	0
				StarAI	2010	19	0
					2012	23	2
					2013	19	0
					2014	24	2
					2015	16	2

5. Datasets

To review and collect datasets already proposed or used in other works, we performed an extensive literature review. The five co-authors investigated publications that appeared in major conferences and journals related to structured machine learning in the past 10 years. An overview of the conferences and journals covered is given in Table 2.

The actual review was performed by either reviewing the accepted papers as linked in the conference schedule or in the corresponding proceedings and journal issues. The papers were scanned for relevant learning tasks involving datasets that are suitable for our benchmarking approach. The following selection criteria were used in order to determine whether a learning task is relevant:

Paper is available One first requirement was to be able to access an electronic version of the paper on the Web. This included PDF versions of the accepted submissions that were made available on the conference website as well as papers provided by the publishers of the corresponding conference proceedings or journals. All considered publications met this criterion.

Availability of the Dataset A major requirement regarding the actual data was its accessibility on the Web to be able to investigate its suitability. In the easiest case downloads and further information were provided on dedicated Web pages. However, frequently datasets were just referred to by name. In such cases we considered a dataset available if we could find an entry via a search engine or in one of the major public machine learning dataset repositories with an unambiguously matching name, file name or description. If datasets were only indirectly referenced by pointing to other publications introducing or using them, we considered them available if we could find a corresponding Web site after (transitively) following and reviewing the referenced papers. Although the portion of datasets actually available varies among the considered literature sources and time, we observed that only a small fraction of approximately 40% of the datasets were accessible, whereas the majority of them stems from benchmark dataset repositories.

Structure of the Dataset Since the main aim of our framework is to provide benchmark scenarios for inductive learning tools working on structured logical representations, we focused on datasets that contain logical relations between single data entries or attributes. Hence, flat datasets mainly describing data with numeric attributes were not considered. However, data that does not comply with this requirement, but could easily be enriched with other structured data, was also further investigated in our review. An example of such a case would be data from clinical trials that could be linked to the Gene Ontology[19] or phenotype ontologies like e.g. the Mammalian Phenotype Ontology[20].

[19]http://geneontology.org/
[20]https://github.com/obophenotype/mammalian-phenotype-ontology

Dataset Size A further requirement was that a candidate dataset should be sufficiently complex in terms of its size. The main aim behind this requirement was to not just provide small toy examples that will show only negligible differences between the benchmarked tools. Ideally, datasets should cover non-synthetic, real world problems to prove the practical applicability of tools obtaining high SML-Bench scores.

Derivable Inductive Learning Problems The last requirement was that the described dataset represents an inductive learning scenario, or that such a scenario could be derived trivially. This means that a supervised machine learning task with positive and (optionally) negative examples is provided or can easily be constructed. In the latter case the dataset must describe certain entities that can be distinguished through relations or data values contained or added to the dataset by means of external data sources. If no example data is given entities from one of those distinguishable sets of entities need to be sampled to serve as positive examples. Negative examples can then be drawn from the remaining distinguished sets. A simple illustration for this procedure is given in case of the Premier League dataset (described below) which provides statistics of soccer players. Here to form one particular learning scenario the positive examples were taken from the set of goal keepers whereas the negative examples were constructed randomly drawing from the set of non-goal keepers, i.e. outfield players. Of course the respective distinctive feature, i.e. a soccer player's field position, need to be removed from the dataset afterwards to avoid trivial solutions.

The review was performed in two rounds: In the *literature review* phase candidate datasets are selected based on their description in the corresponding paper or after briefly checking the actual data. The publications were then marked as either *not*, *maybe*, or *likely* containing suitable datasets. In the *candidate review* round all papers *maybe* or *likely* containing suitable datasets were examined in depth. Here we tried to download the datasets and assessed them manually exploring the raw data (e.g. in a text editor or via SQL queries after loading it into a database).

Overall 6 890 publications were reviewed and 160 candidate datasets selected. From the datasets that were found and could be used for our framework, data conversions and adaptions were performed to work with all tools, if necessary. Besides common and simple formats like CSV or relational databases, data was also provided in special file formats like the *Chemical Table file (CTfile)*[21]. If not available, dedicated parsers and converters had to be written to transform such data into the different KR language formats. Apart from the conversion of the actual data, metadata was added. This additional information comprised TBox axioms in case of OWL background knowledge, and *mode declarations* which had to be added for each Prolog-based learning system. In addition to these efforts, usual testing cycles were performed to check whether the (constructed) learning problem contains enough and consistent information for inductive learning.

As of March 2017, 9 out of 78 datasets were converted and integrated in SML-Bench. Due to the high effort to prepare benchmarks of good quality, including the configuration and verification in the participating inductive learning programs, this is an ongoing task for which we also acknowledge support from the community. The datasets were then labelled with the initial version tag *v0.1* and added to our learning task repository. After internal reviews and discussions or external user feedback this version number can be increased to allow a unique reference to the dataset and represent its maturity.

Apart from conference and journal publications, we also investigated public machine learning dataset repositories mentioned in the reviewed literature. The investigated repositories are summarised in Table 3. If possible, we pre-filtered the repository to classification datasets, ignoring regression or clustering use cases. However, we also examined all datasets that were not grouped into any of those categories or could not be pre-filtered. Most of the repositories provide data that is mainly numeric and lacks a sufficiently deep structure. More deeply structured graph or network datasets, however, usually only provide one kind of edge which would translate into having, e.g. just one Prolog predicate or OWL property in the whole knowledge base, and thus would render those use cases unrealistic for structured machine learning approaches. Another criterion that was rarely met

[21]`http://accelrys.com/products/collaborative-science/biovia-draw/ctfile-no-fee.html`

Table 3

Surveyed benchmark dataset repositories with their number of datasets (#D) as of March 2017, and the number of candidates (#C) that could be used in our framework.

Repository	#D	#C
UCI Machine Learning Repository[22]	349	2
StatLib Datasets Archive[23]	104	0
Stanford Large Network Dataset Collection[24]	103	0
LIBSVM Data[25]	100	0
Relational Dataset Repository[26]	66	4
DL-Learner example datasets[27]	26	5
Mulan Datasets[28]	26	0
Delve Datasets[29]	18	0
IDA Benchmark Repository[30]	13	0

was to have a dataset that is sufficiently large to serve as a benchmark task. For example, even though the DL-Learner repository comes with a lot of learning scenarios that are perfectly tailored for the inductive learning tasks on structured data, most of them would be too simple to distinguish a field of state-of-the-art structured machine learning algorithms. Thus, overall only a very small portion of datasets were considered suitable for our benchmarking purposes.

An overview of datasets that are part of the SML-Bench framework is given in Table 4 and 5.

The datasets *Lymphography*, *Mammographic*, *Pyrimidine* and *Suramin* are rather small in terms of their instance and schema data. They have a very low expressivity and mainly differ in their number of examples with Suramin having the fewest (16), followed by Pyrimidine (40), Lymphography (148), and Mammographic (961).

Mutagenesis, *Hepatitis* and *Carcinogenesis* can be considered as 'medium size' datasets w.r.t our benchmarking repository. While the OWL representation of the Mutagenesis dataset also shares the very simple DL family $\mathcal{AL}(D)$, Hepatitis and Carcinogenesis are more complex. However, with $\mathcal{ALC}(D)$ being the most expressive DL used, they can still be considered simple. In terms of their example sets, Mutagenesis provides the smallest number (84), followed by Carcinogenesis (298) and Hepatitis (500).

The most complex dataset w.r.t. the expressivity of its OWL representation is NCTRER. In size, however, it can also be considered medium, be it in terms of schema and instance data, or w.r.t. the number of examples.

Table 4

Overview of the datasets that are part of the SML-Bench framework

Dataset	Description
Carcinogenesis	Prediction of carcinogenic drugs
Hepatitis	Prediction of the Hepatitis type based on patient data
Lymphography	Prediction of diagnosis class based on lymphography patient data
Mammographic	Prediction of breast cancer severity based on screening data
Mutagenesis	Prediction of the mutagenicity of chemical compounds
NCTRER	Prediction of a molecule's estrogen receptor binding activity
Premier League	Find a predictive description of goal keepers based on player statistics in soccer matches
Pyrimidine	Prediction of the inhibition activity of pyrimidines and the DHFR enzyme
Suramin	Find a predictive description of suramin analogues for cancer treatment

Table 5

Overview of the OWL versions of the datasets that are part of SML-Bench with their number of axioms (#A), classes (#C), object properties (#O), datatype properties (#D) and expressivity (Expr.)

Dataset	#A	#C	#O	#D	Expr.
Carcinogenesis	74,566	142	4	15	$\mathcal{ALC}(D)$
Hepatitis	73,114	14	5	12	$\mathcal{ALE}(D)$
Lymphography	2,187	53	0	0	$\mathcal{AL}$
Mammographic	6,808	19	3	2	$\mathcal{AL}(D)$
Mutagenesis	62,066	86	5	6	$\mathcal{AL}(D)$
NCTRER	92,861	37	9	50	$\mathcal{ALCI}(D)$
Prem. League	214,566	10	14	202	$\mathcal{ALEH}(D)$
Pyrimidine	2,006	1	0	27	$\mathcal{AL}(D)$
Suramin	13,506	46	3	1	$\mathcal{AL}(D)$

[22]https://archive.ics.uci.edu/ml/datasets.html

[23]http://lib.stat.cmu.edu/datasets/

[24]http://snap.stanford.edu/data/index.html

[25]https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

[26]https://relational.fit.cvut.cz/

[27]https://github.com/SmartDataAnalytics/DL-Learner (*test* and *examples* directories)

[28]http://mulan.sourceforge.net/datasets-mlc.html

[29]http://www.cs.toronto.edu/~delve/data/datasets.html

[30]http://www.raetschlab.org/Members/raetsch/benchmark

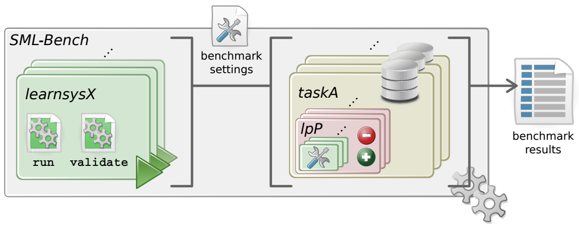

Fig. 1. Overview of the SML-Bench framework. Learning problems (*lpP*, red) are defined on a learning task (*taskA*, yellowish) and contain positive/negative examples and optional learning system configurations. An overall benchmark configuration defines which learning systems (*learnsysX*, green) to run on which learning problem to produce benchmark results.

The biggest dataset we provide is Premier League. It provides a lot of different statistics which are expressed through an extensive (though still simple) schema and comprehensive instance data.

Further information about the datasets (like their origin, size, etc.) can be found in the dataset description files in the SML-Bench GitHub repository. The dataset descriptions are provided as RDF data and can be found in the respective knowledge representation language directory for each learning task[31].

6. SML-Bench Framework

With SML-Bench our aim is to provide a framework which is open and extensible but already comes with predefined benchmark scenarios and presets for relevant tools, thus being ready for use. The core framework is developed in the Java programming language and is intended to be used via a command line interface. However, the system can easily be extended to support graphical user interfaces. SML-Bench is provided as free software and accessible on the Web[32].

Architecture The overall architecture of SML-Bench is shown in Figure 1. The framework's main building blocks are the tools to execute during a benchmark run and the benchmark scenarios. SML-Bench provides means to connect a set of inductive learning tools with such scenarios to run the evaluation on. This overall setting is held in a benchmark configuration and the framework will take care of providing the tools with the required data, performing the benchmark and collecting the results.

To support a wide range of tools and the introduction of own inductive learning implementations the SML-Bench framework follows a lightweight extensibility approach. Based on the relations between benchmark scenarios, their background knowledge, utilised KR languages, and benchmarked inductive learning systems we define some conventions which allow the extension of the framework with new use cases and tools without any changes in the code base or further wiring.

Benchmark scenarios To better structure scenarios and allow different benchmark variations based on the same data we distinguish between *learning tasks* and *learning problems*. Learning tasks define the actual background knowledge the benchmark is run on for learning problems. Learning problems are thus learning task-specific and comprise a set of positive and (optionally) negative examples, as well as optional tool settings dedicated to the given example declarations (cf. Figure 1). Accordingly, varying example constellations or tool configurations are realised as separate learning problems.

In our 'convention over configuration' approach the files containing the background knowledge for a learning task A, given in a knowledge representation language L, are expected to reside in the directory path *learningtasks/**A**/**L**/data/* (relative to the framework's root directory). Examples for L, in use already, are *owl* and *prolog*. If additional tool-specific data is required (as in case of the Prolog-based tools which usually require particular mode declarations), e.g. for a tool X, this should be put into the directory *learningtasks/A/L/data/**X**/*. Data might be spread across multiple files which are all read and merged during a benchmark run.

An individual learning problem P can be defined adding a file containing positive examples and an optional file containing negative examples to a directory named *learningtasks/A/L/lp/**P**/*. A learning problem might also comprise tool-specific configurations which are put into a file named after the respective tool with the file suffix *.conf*, e.g. *learningtasks/A/L/lp/P/**X.conf***.

[31] E.g. https://github.com/SmartDataAnalytics/SML-Bench/blob/develop/learningtasks/carcinogenesis/owl/dataset.ttl for the OWL version of the Carcinogenesis dataset

[32] https://github.com/SmartDataAnalytics/SML-Bench

Benchmarked Tools A similar approach is followed to integrate inductive learning tools into SML-Bench. To make a tool X available to the benchmark framework, a corresponding directory has to be created at *learningsystems/**X**/*. For a given *learning system* under assessment we are mainly interested in two things: 1) the learned rule or class expression and 2) a measure of how well this rule or expression performs on the provided examples. Accordingly, the benchmark process is divided into two phases: 1) the *training phase* and 2) the *validation phase*. In the training phase the learning system generates the rule or class expression for a particular learning problem, which is then assessed in the validation phase. Whereas the output of phase 1 might be a tool-specific representation of the learned description, the validation output has to follow a fixed pattern, quantifying the number of true positive, false positive, true negative and false negative examples covered. Since the integration of all these particularities into the core framework would render it inflexible and hardly extensible, we rely on a wrapper-based interface to the respective learning systems: For each phase a dedicated executable has to be provided. One executable runs the learning system to generate the rules or class expressions which are then assessed by a validation executable producing the standardised output. These executables have to be named *run* and *validate*, respectively, and can be written in any programming language (cf. Figure 1).

To illustrate the extensibility we consider an imaginary learning system with the main executable *systemx* which takes the file paths to the OWL background knowledge, positive and negative examples as input and generates an OWL class expression written to an output file specified as last argument. A possible *run* shell script adapting this system to SML-Bench is sketched in Listing 1.

Listing 1: Example run script

```bash
#!/bin/bash

# Makes helper functions like read_config available
# (not part of SML-Bench -- must be provided)
source myhelperfunctions.sh

# Reads the benchmark configuration properties file given
# to each run executable.
# All properties (like e.g. filename.workdir) will then be
# made available as environment variables (with dots
# replaced by double underscores to form valid variable
# names). The main settings of interest are
# filename.workdir, filename.pos, filename.neg,
# filename.output and the current learning task.
read_config

dataset_dir="${filename__workdir}/learningtasks/\
${learningtask}/owl/data/"
# Concatenates all OWL files to one single file
cat ${dataset_dir}*.owl > bg_data.owl

./systemx bg_data.owl $filename__pos $filename__neg \
    $filename__output

exit 0
```

In this example the result written to the output file is a (tool-specific) class expression string like e.g. *hasStructure some (not (Amine))*. In the next step the *validate* script will be invoked which parses this string, again reads files containing positive and negative examples, and feeds the background knowledge files into an OWL reasoner. This OWL reasoner will then be used to assess the result class expression in terms of provided positive examples being an instance of it (true positives, tp), positive examples not not being an instance of it (false negatives, fn), negative examples being an instance of it (false positives, fp), and negative examples not being an instance of it (true negatives, tn). The expected output of this script will be a file with the counts for the respective classification category as shown in Listing 2.

Listing 2: Example validation result file

```
tp: 33
fp: 2
tn: 35
fn: 0
```

Based on those numbers measures like accuracy, F-score etc. are computed.

Benchmark settings To generate a custom benchmark on a selection of tools and learning problems, a global configuration has to be provided defining which tools to run, which learning problems to tackle, and optionally, additional benchmark-specific tool configurations. The framework then executes the *run* and *validate* executables with the corresponding configurations of all selected tools. SML-Bench supports arbitrary train-test splits, n-fold cross validation, as well as running the training and validation on the whole set of examples[33]. The actual execution can be performed in parallel threads or sequentially. A simple benchmark configuration snippet is shown in Listing 3.

Listing 3: Example configuration of the SML-Bench benchmark runner

```
learningsystems = aleph, dllearner, progol, progolem
learningsystems.aleph.noise = 3
learningsystems.progol.noise = 3
learningsystems.progolem.positive_example_inflation=2

scenarios = mammographic/1, mutagenesis/42

framework.crossValidationFolds = 10
framework.maxExecutionTime = 1800
framework.threads = 10
```

SML-Bench supports the generation of semantic descriptions of a benchmark setup based on the MEX vocabulary [7]. Such descriptions do not only comprise general configuration issues as shown in Listing 3 but cover all the details to comprehend the benchmark settings in detail. This includes detailed specifications of the tools executed together with their runtime configurations, details about the data used in the benchmark and the actual evaluation results.

Available learning systems In its current state SML-Bench supports eight inductive learning tools, collected during our literature review. A tool was introduced into SML-Bench as learning system if it implements a published inductive learning algorithm, if it is freely available and sufficiently documented.

The oldest of the available learning systems is the classic ILP tool *Golem* [22] which was published in the year 1990, implementing a *Relative Least General Generalisations*-based induction approach. Golem supports a Prolog-based knowledge representation language. Another, slightly more recent ILP tool called *Progol* [21] uses inverse entailment to derive covering clauses based on examples and background knowledge given as Prolog-like logic programs. An ILP tool completely implemented in Prolog is *Aleph* which supports a number of ILP algorithms. Similarly, the *General Inductive Logic Programming System (GILPS)* comprises several Prolog programs realising different inductive learning approaches. FuncLog [25], one tool of the GILPS collection, is specialised in learning on *Head Output Connected* learning problems. Besides this, the tools *TopLog* [24] and *ProGolem* [23] which are based on *Top Directed Hypothesis Derivation* and *Asymmetric Relative Minimal Generalisations*, respectively, are also part of GILPS and supported in SML-Bench.

On the description logics-based knowledge representation field, several algorithms are integrated via one tool: *DL-Learner* [16] is a framework for inductive learning on RDF and OWL-based background knowledge. It supports a wide range of algorithms, including refinement operator based algorithms and evolutionary inspired approaches, as well as different OWL profiles.

In terms of *Statistical Relational Learning* (SRL) tools, we considered *RapidMiner*[34] and *TreeLiker*[35] [13]. Unfortunately we were not able to integrate RapidMiner since its server component imposed requirements that were not fulfilled in our overall workflow[36]. However, we provide experimental support for the TreeLiker tool, which also works on background knowledge expressed in a Prolog-like syntax and contains implementations for different SRL algorithms.

7. Evaluation

To give a general impression of our framework, we ran SML-Bench with all provided learning systems on all available learning problems, excluding Suramin due to its small number of examples. We applied 10-fold cross validation and executed all learning systems in their default settings except the DL-Learner running OCEL which requires a noise value to be set to allow a certain number of misclassifications. Thus, we set the *noisePercentage* parameter to 30. Leaving the learning systems unconfigured will not show their optimal performance, but will rather reflect whether a tool performs well 'out-of-the-box' and whether the executed algorithms fit the particular learning scenar-

[33]Though uncommon in machine learning, this was a requirement in a 3rd party use case for a separate feature extraction phase over the example set.

[34]https://rapidminer.com

[35]http://ida.felk.cvut.cz/treeliker/TreeLiker.html

[36]See https://github.com/SmartDataAnalytics/SML-Bench/issues/14 for more details

ios. This also means that highly specialized algorithms might not work well on all of the learning scenarios, or even, that our learning problems do not have certain, expected characteristics an algorithm requires to function well. Following this approach our main aim is to assess whether SML-Bench can provide a meaningful benchmarking environment. To consider the provided set of benchmarking tasks meaningful we would expect to see that 1) not all learning problems can be 'solved' easily in default settings 2) the learning problems are able to distinguish the field of competitors, i.e. that they point out certain strengths and weaknesses of the learning systems under assessment 3) the learning problems are diverse enough to address different strengths and weaknesses of the learning systems under assessment.

In the case of DL-Learner and TreeLiker we executed a set of available algorithms introduced in the following. The *OWL Class Expression Learner* (OCEL) algorithm, which is part of the DL-Learner, is a refinement operator-based learning algorithm using heuristics guiding the search. An evolution of OCEL which is more biased towards short and human readable concepts is the *Class Expression Learning for Ontology Engineering* (CELOE) algorithm [17]. A third implementation provided by the DL-Learner framework is the *EL Tree Learner* (ELTL) algorithm which is restricted to OWL EL as target concept language.

The TreeLiker tool can be configured to utilize a *block-wise construction of tree-like relational features* (RelF) [15], a *hierarchical feature construction* (HiFi) [14], or a *Gaussian Logic-based* algorithm (Poly) [12] for classification. These three algorithms can also be run in a *grounding-counting setting* (GC) considering the number of examples covered by a generated feature during learning. Since the TreeLiker works on a Prolog-like knowledge representation language that does not support certain Prolog expressions, we could not assess it on the Mutagenesis and NCTRER datasets.

We set an overall maximum execution time of 300 seconds and executed all tools sequentially. The benchmark was performed on a machine with 2 Intel Xeon 'Broadwell' CPUs with 8 cores running at 2.1 GHz with 128 GB of RAM. A benchmark description based on the MEX RDF vocabulary can be found at `http://sml-bench.sda.tech/benchmark_results/march2017/`. The benchmark results are summarised in Table 6 and Table 7. Besides their average accuracies and F_1-scores we also report when nothing (or just the trivial solution listing all the input examples) could be learned (*no results*), when learning systems could not finish within the given 300 seconds (*time-out*), or when learning systems ran out of memory (*out of mem.*).

One first observation we made is that we seem not to have a suitable learning scenario which would benefit from FuncLog's specialization in learning with head output connected predicates. Since it did not return learned rules on any of the learning problems we did not list FuncLog in Table 6 and Table 7.

Another observation is that Aleph, CELOE and OCEL already provide default settings which work well on the learning problems and lead to very good results on mutagenesis/42, premierleague/1 and pyrimidine/1.

Taking into consideration that Golem was implemented in the 1990s one possible explanation for its lower performance could be that the default settings reflect hardware expectations in terms of available memory and computing power that are now superseded. This might also apply to constants defined in Golem's source code. Hence, adjusting settings to current hardware capabilities might make a considerable difference here. A similar argumentation might apply for Progol. Besides this Golem's mode declarations do not provide means to express explicitly which predicates should appear in the head of a learned rule. This might be an explanation for some of the results that do not provide a description for the given examples, at all, as in the case of nctrer/1 where we observed learned rules like *bound_atom(A,B) :- first_bound_atom(A,atom_232_2)* that should actually characterize molecules, i.e. positive examples like *molecule(molecule13)*.

Progol and ProGolem appear to be overly curtailed by the restricted execution time. In those cases, the algorithm itself may be very suitable for the learning problems but increasing the time limit further would lead to prohibitive runtimes of the evaluation scenario. Currently, the maximum runtime is approximately 100 hours (10 folds $\times$ 8 learning problems $\times$ 15 configured learning systems $\times$ 5 minutes).

In their default settings, the GILPS tools ProGolem and TopLog often returned identical results. Even though we can only report a low performance

Table 6

Evaluation results of an SML-Bench benchmark run. All tools were run with a maximum execution time of 5 minutes. Reported are the **average accuracy** and its standard deviation of 10-fold cross validation.

Learning problem	Aleph	DL-Learner (CELOE)	DL-Learner (OCEL)	DL-Learner (ELTL)	Golem	Progol	ProGolem
carcinog./1	0.48 ± 0.10	$\mathbf{0.55 \pm 0.02}$	*no results*	$\mathbf{0.55 \pm 0.02}$	*no results*	0.49 ± 0.06	*timeout*
hepatitis/1	$\mathbf{0.67 \pm 0.05}$	0.47 ± 0.05	0.66 ± 0.14	0.41 ± 0.01	0.59 ± 0.01	*no results*	0.29 ± 0.10
lymphogr./1	$\mathbf{0.83 \pm 0.10}$	0.83 ± 0.11	0.73 ± 0.12	0.54 ± 0.03	0.40 ± 0.11	0.79 ± 0.09	0.28 ± 0.16
mammogr./1	0.65 ± 0.04	0.49 ± 0.02	$\mathbf{0.82 \pm 0.05}$	0.46 ± 0.01	0.54 ± 0.00	*timeout*	*timeout*
mutag./42	0.72 ± 0.25	$\mathbf{0.94 \pm 0.13}$	0.53 ± 0.29	0.30 ± 0.07	0.42 ± 0.25	*no results*	*timeout*
nctrer/1	0.72 ± 0.14	0.59 ± 0.02	$\mathbf{0.81 \pm 0.09}$	0.58 ± 0.02	0.41 ± 0.01	*no results*	0.00 ± 0.00
prem.leag./1	0.95 ± 0.09	$\mathbf{0.99 \pm 0.04}$	0.85 ± 0.10	0.49 ± 0.02	*out of mem.*	*no results*	0.00 ± 0.00
pyrimidine/1	$\mathbf{0.95 \pm 0.16}$	0.83 ± 0.17	0.85 ± 0.24	*no results*	0.15 ± 0.21	*no results*	0.35 ± 0.32

Learning problem	TopLog	TreeLiker (HiFi)	TreeLiker (RelF)	TreeLiker (Poly)	TreeLiker (HiFi GC)	TreeLiker (RelF GC)	TreeLiker (Poly GC)
carcinog./1	0.40 ± 0.12	0.38 ± 0.12	0.38 ± 0.12	0.38 ± 0.12	0.43 ± 0.11	0.43 ± 0.11	0.38 ± 0.12
hepatitis/1	0.18 ± 0.04	0.46 ± 0.05	0.46 ± 0.05	0.46 ± 0.05	0.49 ± 0.16	0.56 ± 0.03	0.46 ± 0.05
lymphogr./1	0.28 ± 0.16	0.29 ± 0.17	0.29 ± 0.17	0.29 ± 0.17	0.52 ± 0.11	0.51 ± 0.09	0.29 ± 0.17
mammogr./1	0.18 ± 0.05	0.22 ± 0.04	0.22 ± 0.04	0.22 ± 0.04	0.54 ± 0.01	0.54 ± 0.01	0.22 ± 0.04
mutag./42	0.23 ± 0.18	—	—	—	—	—	—
nctrer/1	0.00 ± 0.00	—	—	—	—	—	—
prem.leag./1	0.00 ± 0.00	*out of mem.*	*out of mem.*	*out of mem.*	*out of mem.*	*out of mem.*	*out of mem.*
pyrimidine/1	0.18 ± 0.26	0.20 ± 0.26	0.20 ± 0.26	0.18 ± 0.26	0.60 ± 0.24	0.60 ± 0.39	0.18 ± 0.26

Table 7

Evaluation results of an SML-Bench benchmark. All tools were run with a max. execution time of 5 minutes. Reported are the **average $\mathbf{F_1}$-score** and its standard deviation of 10-fold cross validation.

Learning problem	Aleph	DL-Learner (CELOE)	DL-Learner (OCEL)	DL-Learner (ELTL)	Golem	Progol	ProGolem
carcinog./1	0.46 ± 0.12	$\mathbf{0.71 \pm 0.01}$	*no results*	$\mathbf{0.71 \pm 0.01}$	*no results*	0.16 ± 0.12	*timeout*
hepatitis/1	0.38 ± 0.12	0.60 ± 0.02	$\mathbf{0.64 \pm 0.07}$	0.58 ± 0.01	0.00 ± 0.00	*no results*	0.32 ± 0.10
lymphogr./1	0.84 ± 0.09	$\mathbf{0.87 \pm 0.07}$	0.76 ± 0.10	0.70 ± 0.03	0.13 ± 0.10	0.79 ± 0.10	0.26 ± 0.18
mammogr./1	0.48 ± 0.08	0.64 ± 0.01	$\mathbf{0.78 \pm 0.08}$	0.63 ± 0.00	0.00 ± 0.00	*timeout*	*timeout*
mutag./42	0.43 ± 0.47	$\mathbf{0.93 \pm 0.14}$	0.29 ± 0.42	0.46 ± 0.08	0.16 ± 0.25	*no results*	*timeout*
nctrer/1	0.71 ± 0.18	0.73 ± 0.02	$\mathbf{0.85 \pm 0.06}$	0.73 ± 0.02	0.00 ± 0.00	*no results*	0.00 ± 0.00
prem.leag./1	0.94 ± 0.11	$\mathbf{0.99 \pm 0.04}$	0.97 ± 0.06	0.66 ± 0.02	*out of mem.*	*no results*	0.00 ± 0.00
pyrimidine/1	$\mathbf{0.90 \pm 0.32}$	0.84 ± 0.15	0.80 ± 0.13	*no results*	0.04 ± 0.13	*no results*	0.33 ± 0.32

Learning problem	TopLog	TreeLiker (HiFi)	TreeLiker (RelF)	TreeLiker (Poly)	TreeLiker (HiFi GC)	TreeLiker (RelF GC)	TreeLiker (Poly GC)
carcinog./1	0.38 ± 0.17	0.39 ± 0.18	0.39 ± 0.18	0.39 ± 0.18	0.16 ± 0.15	0.16 ± 0.15	0.39 ± 0.18
hepatitis/1	0.21 ± 0.08	0.48 ± 0.10	0.48 ± 0.10	0.48 ± 0.10	0.26 ± 0.22	0.17 ± 0.18	0.48 ± 0.10
lymphogr./1	0.26 ± 0.18	0.24 ± 0.19	0.24 ± 0.19	0.23 ± 0.19	0.31 ± 0.23	0.20 ± 0.22	0.23 ± 0.19
mammogr./1	0.24 ± 0.07	0.24 ± 0.08	0.24 ± 0.08	0.24 ± 0.08	0.02 ± 0.03	0.02 ± 0.03	0.24 ± 0.08
mutag./42	0.23 ± 0.25	—	—	—	—	—	—
nctrer/1	0.00 ± 0.00	—	—	—	—	—	—
prem.leag./1	0.00 ± 0.00	*out of mem.*	*out of mem.*	*out of mem.*	*out of mem.*	*out of mem.*	*out of mem.*
pyrimidine/1	0.20 ± 0.26	0.20 ± 0.26	0.20 ± 0.26	0.20 ± 0.26	0.68 ± 0.24	0.50 ± 0.45	0.20 ± 0.26

on all the learning problems, the authors of ProGolem and TopLog published experiments showing much better results on Carcinogenesis, Pyrimidine and Mutagenesis [24,23]. This might emphasize that a proper configuration substantially impacts the tool's performance, or suggest that differing versions of the respective datasets were in use.

For the TreeLiker algorithms we can also observe that different settings might give identical results. The low performance can be attributed to the fact that TreeLiker is a collection of feature construction algorithms and to the way we use it in our benchmark framework. Since the TreeLiker algorithms usually produce a high number of features we currently only consider the best one for our evaluation. This might not fully exploit TreeLiker's capabilities and we are in contact with one of the tool authors to improve this.

Even though the premierleague/1 learning problem is large in terms of background knowledge with more than 200 thousand axioms, Aleph was able to learn almost perfect results. For the TreeLiker we gradually increased the maximum available JVM heap size up to 10 GB. Increasing it even further might also give results for its algorithms. However, since other Java implementations could generate results with 2 GB of available maximum heap size, we stopped there.

Overall, the evaluation supports our initial expectations. However, we also have to admit that some of the learning systems need to be adjusted properly to provide competitive performances. This will be discussed in the following.

8. Discussion

With SML-Bench we built a benchmarking framework that is extensible and comes with a set of initial scenarios to evaluate arbitrary inductive learning tools. As shown in the previous section, the provided learning problems are able to discriminate the performance of different learning systems but are not complete in the sense that we are lacking some datasets that are tailored for particular capabilities of certain tools (in particular FuncLog). We also believe – and verified this in some cases manually – that most of the results can be improved by spending more effort in configuring the learning systems, hence generating more competitive results. We already tried to get in contact with the tool authors to support this, but only got a reply from one of the TreeLiker developers. In the future, we may allow an explicit parameter tuning phase, e.g. via nested cross validation or explicit tuning examples, in our benchmark for systems which are capable of this functionality. Apart from the issues revolving around system configuration, the literature survey and the actually converted datasets have shown that datatype properties are widely used in many learning scenarios. However, the tools currently do not fully exploit this part and focus more on the structured components. In this sense, the tools would benefit more from deeper structures in the datasets. To this end, we will work on further enriching datasets in this direction if possible. Of course doing so requires considerable effort and extensive domain knowledge (e.g. in chemistry or genetics). Through the community feedback we obtain, we will continuously extend and refine the learning problem library. SML-Bench is part of a funded research programme and benchmarking challenges will be presented in a series of (yet to be finally determined) venues. We will use those as a feedback channel.

A further issue that needs to be discussed is the representation of knowledge in different KR languages. Most of the design decisions for the dataset conversions were made individually, since an automatic conversion would not yield a satisfactory modelling result exploiting the strengths of Prolog or different OWL profiles even in cases when it is theoretically possible. This can potentially lead to a bias since particular modelling choices may lead to different solutions provided by the tools. While we acknowledge this problem, we do not see a straightforward solution and also believe that to some extent this could spur some competition in terms of finding appropriate KR languages or dialects to support in inductive learning.

In our opinion, the availability of SML-Bench will improve the state of the art for symbolic machine learning from expressive background knowledge in the next years. While many efforts in this field date back to the early 90s, only in the past years the availability of data has increased significantly. However, it was still challenging for individual researchers or small groups to perform a comprehensive benchmark. We believe to have closed this gap, which could in turn lead to similar improvements as we have seen for question answering, link discovery and query performance for RDF.

9. Conclusion and Outlook

In this paper, we have presented SML-Bench – a benchmarking framework for structured machine learning. We performed a systematic literature survey to obtain relevant benchmark datasets. Overall, we analysed 6 890 papers, which led to 160 candidate learning problems. For 9 of those, we converted them across the used KR languages and set them up for all learning systems. Currently, 8 learning systems are integrated with two further inclusion requests in the works. A first analysis presented here has identified some shortcomings of individual tools. Generally, we believe that a mature research area requires a benchmark to evolve further. In particular, we want to contribute to bringing the Semantic Web and machine learning areas close together. We also aim to reduce the boundaries of knowledge representation languages as well as the research communities behind them. We further envision that SML-Bench could in the future evolve into a central hub for comparing suggested tool settings, learning problems, and performances.

We will perform further analysis and regular benchmarking runs in the scope of the HOBBIT project[37] which will fund this benchmarking activity until end of 2018 after which it will be taken over by the HOBBIT association. SML-Bench based challenges are planned in further workshops, e.g. the Know@LOD workshop to which we contributed for the past 5 years. In the future, it is likely that we will support further languages, e.g. full first order logic based systems, combinations of rules and description logics as well as fuzzy and probabilistic description logics and statistical relational learning systems.

Another direction for future work is the integration of means to import MEX machine learning experiment descriptions to generate benchmark configurations. In combination with our MEX export function this would allow to load experiments from other researchers or share own benchmark settings.

[37] http://project-hobbit.eu/

References

[1] K. Albayraktaroglu, A. Jaleel, X. Wu, M. Franklin, B. Jacob, C.-W. Tseng, and D. Yeung. Biobench: A benchmark suite of bioinformatics applications. In *Performance Analysis of Systems and Software, 2005. ISPASS 2005. IEEE International Symposium on*, pages 2–9. IEEE, 2005.

[2] R. Angles, P. Boncz, J. Larriba-Pey, I. Fundulaki, T. Neumann, O. Erling, P. Neubauer, N. Martinez-Bazan, V. Kotsev, and I. Toma. The linked data benchmark council: a graph and rdf industry benchmarking effort. *ACM SIGMOD Record*, 43(1):27–31, 2014.

[3] C. Baru, M. Bhandarkar, R. Nambiar, M. Poess, and T. Rabl. Setting the direction for big data benchmark standards. In *Selected Topics in Performance Evaluation and Benchmarking*, pages 197–208. Springer, 2012.

[4] C. Baru, M. Bhandarkar, R. Nambiar, M. Poess, and T. Rabl. Benchmarking big data systems and the bigdata top100 list. *Big Data*, 1(1):60–64, 2013.

[5] C. Blake and C. J. Merz. {UCI} repository of machine learning databases. 1998.

[6] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[7] D. Esteves, D. Moussallem, C. B. Neto, T. Soru, R. Usbeck, M. Ackermann, and J. Lehmann. Mex vocabulary: A lightweight interchange format for machine learning experiments. In *11th International Conference on Semantic Systems (SEMANTiCS 2015), 15-17 September 2015, Vienna, Austria*, 2015.

[8] G. Fox, S. Jha, J. Qiu, S. Ekanazake, and A. Luckow. Towards a comprehensive set of big data benchmarks. *Big Data and High Performance Computing*, 26:47, 2015.

[9] W. Gao, Y. Zhu, Z. Jia, C. Luo, L. Wang, Z. Li, J. Zhan, Y. Qi, Y. He, S. Gong, et al. Bigdatabench: a big data benchmark suite from web search engines. *arXiv preprint arXiv:1307.0320*, 2013.

[10] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen. Bigbench: towards an industry standard benchmark for big data analytics. In *Proceedings of the 2013 ACM SIGMOD international conference on Management of data*, pages 1197–1208. ACM, 2013.

[11] R. D. King. Statlog databases. *Department of Statistics and Modelling Science, University of Strathclyde, Glasgow, UK*, 1992.

[12] O. Kuželka, A. Szabóová, M. Holec, and F. Železný. Gaussian logic for predictive classification. In *ECML/PKDD 2011: European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2011.

[13] O. Kuželka, A. Szabóová, and F. Železný. Relational learning with polynomials. In *24th International Conference on Tools with Artificial Intelligence*, 2012.

[14] O. Kuželka and F. Železný. Hifi: Tractable propositionalization through hierarchical feature construction. In *18th International Conference on Inductive Logic Programming*, 2008.

[15] O. Kuželka and F. Železný. Block-wise construction of tree-like relational features with monotone reducibility and redundancy. *Machine Learning*, 83(2):163–192, 2011.

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

[17] J. Lehmann, S. Auer, L. Bühmann, and S. Tramp. Class expression learning for ontology engineering. *Journal of Web Semantics*, 9:71 – 81, 2011.

[18] M. Li, J. Tan, Y. Wang, L. Zhang, and V. Salapura. Sparkbench: A comprehensive benchmarking suite for in memory data analytic platform spark. In *Proceedings of the 12th ACM International Conference on Computing Frontiers*, page 53. ACM, 2015.

[19] V. Lopez, C. Unger, P. Cimiano, and E. Motta. Evaluating question answering over linked data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 21:3–13, 2013.

[20] Z. Ming, C. Luo, W. Gao, R. Han, Q. Yang, L. Wang, and J. Zhan. Bdgs: A scalable big data generator suite in big data benchmarking. In *Advancing Big Data Benchmarks*, pages 138–154. Springer, 2013.

[21] S. Muggleton. Inverse entailment and progol. *New Generation Computing*, 13(3):245–286, Dec. 1995.

[22] S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the First Conference on Algorithmic Learning Theory*, 1990.

[23] S. Muggleton, J. C. A. Santos, and A. Tamaddoni-Nezhad. Progolem: A system based on relative minimal generalisation. In *Proceedings of the 19th International Conference on Inductive Logic Programming*, 2010.

[24] S. Muggleton and J. C. A. S. A. Tamaddoni-Nezhad. Toplog: Ilp using a logic program declarative bias. In *24th International Conference on Logic Programming*, 2008.

[25] J. C. A. Santos, A. Tamaddoni-Nezhad, and S. Muggleton. An ilp system for learning head output connected predicates. In *Proceedings of the 14th Portuguese Conference on Artificial Intelligence*, 2009.

[26] M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009.

[27] P. Vlachos and M. Meyer. Statlib datasets archive. *URL http://lib. stat. cmu. edu/datasets*, 2005.

[28] E. J. Weyuker and F. I. Vokolos. Experience with performance testing of software systems: Issues, an approach, and case study. *IEEE transactions on software engineering*, 26(12):1147–1156, 2000.