# ML-Schema: An interchangeable format for description of machine learning experiments

Gustavo Correa Publio [a,*], Agnieszka Ławrynowicz [b], Larisa Soldatova [c], Panče Panov [d,i], Diego Esteves [e,h], Joaquin Vanschoren [f], Tommaso Soru [g]

[a] *Institut für Informatik, AKSW Group, Universität Leipzig, Germany*
*E-mail: gustavo.publio@informatik.uni-leipzig.de*
[b] *Faculty of Computing, Poznan University of Technology, Poland*
*E-mail: agnieszka.lawrynowicz@cs.put.poznan.pl*
[c] *Department of Computing, Goldsmiths, University of London,United Kingdom*
*E-mail: l.soldatova@gold.ac.uk*
[d] *Department of Knowledge Technologies, Jožef Stefan Institute,Slovenia*
*E-mail: pance.panov@ijs.si*
[e] *SDA Research, University of Bonn,Germany*
*E-mail: diego.esteves@farfetch.com*
[f] *Mathematics and Computer Science, Eindhoven University of Technology,Netherlands*
*E-mail: j.vanschoren@tue.nl*
[g] *Research Group, Semantic Integration Ltd.,United Kingdom*
*E-mail: tom@tommaso-soru.it*
[h] *Search and Discovery, Farfetch,Portugal*
*E-mail: diego.esteves@farfetch.com*
[i] *Jožef Stefan International Postgraduate School, Ljubljana, Slovenia*
*E-mail: pance.panov@ijs.si*

**Abstract.** In this paper, we present the ML-Schema, proposed by the W3C Machine Learning Schema Community Group. ML-Schema is a top-level ontology that provides a set of classes, properties, and restrictions for representing and interchanging information on machine learning algorithms, datasets, and experiments. ML-Schema, a canonical format, resulted of more than seven years of experience of different research institutions. We discuss the main challenges in the development of ML-Schema, which have been to align existing machine learning ontologies and other relevant representations designed for a range of particular purposes following sometimes incompatible design principles, resulting in different not easily interoperable structures. The resulting ML-Schema can now be easily extended and specialized allowing to map other more domain-specific ontologies developed in the area of machine learning and data mining.

Keywords: ontology, data interchange standard, machine learning

## 1. Introduction

Machine learning (ML) experiments are complex studies involving many steps and iterations requiring expert knowledge. Ensuring that ML research outcomes are properly comparable, understandable, interpretable, reusable and reproducible is a challenge that many proposals, such as *Wings* [1], *OpenTox* [2] and *MyExperiment* [3], have tried to address. Nevertheless, each of them deals with a set of specific scenarios

*Corresponding author. E-mail: gustavo.publio@informatik.uni-leipzig.de.

and fails to address a broader, generic approach in the context of reproducible and reusable science.

Ontologies, as formal machine-readable knowledge representations, have the potential to help achieve this goal. An ontology formally defines essential concepts, their properties, and relevant axioms pertinent to a particular area of interest [4].

In the last decade, several ontologies have been proposed to formally represent and model the area of machine learning and data mining. *Onto-DM* (an Ontology of Data Mining) was designed to provide generic representations of principle entities in the area of data mining [5, 6]. *DMOP* (Data Mining OPtimization ontology) has been developed to support meta-mining, i.e. meta-learning from complete ML processes [7]. *Exposé* has been designed to describe and reason about ML experiments [8]. It underpins *OpenML*[1] [9], a collaborative meta-learning platform for machine learning that embodies the concept of experimental databases [10]. Finally, the *MEX Vocabulary* (composed of three modules: mex-core, mex-algo and mex-perf) aims to tackle the problem of managing ML outcomes and sharing provenance information, particularly on the basic ML iterations, in a lightweight format [11].

The development of these ML ontologies is a significant step towards ensuring unambiguous interpretability and reproducibility of ML experiments. However, none of the existing ontologies fully covers the area of machine learning and supports all the needs for the representation and encoding of ML experiments.

Instead of the development of a comprehensive general-purpose ML ontology, here we propose a more practical and flexible approach that involves the development of ML-Schema – Machine Learning Schema (MLS) – for mapping the existing ML ontologies and to support a variety of useful extensions. To achieve this ambitious goal, in September 2015 developers of several ML ontologies (OntoDM, DMOP, Expose and MEX) formed a W3C Community Group[2]. The development of MLS has been initiated as an attempt to prevent a proliferation of incompatible ML ontologies and to increase interoperability among existing ones. The *MLS Community Group* (MLS-CG) is an open-source community currently comprehending over 50 international researchers and industry representatives.

The main challenge in the development of MLS is to align existing ML ontologies and other relevant representations designed for a range of particular purposes following sometimes incompatible design principles, resulting in different not easily interoperable structures. Moreover, ML experiments are executed on different ML software platforms; each of those having specific conceptualization or schema for representing data and meta-data about the experiment.

To address the challenge, the members of the MLS-CG identified and aligned them with the related ontologies and vocabularies. The schema is focusing on the representation of the algorithms, the machine learning tasks they address, their implementations and executions, as well as inputs (e.g., data), outputs (e.g., models), and performances. The schema also defines a relationship between machine learning algorithms and their single executions (runs), experiments and studies encompassing them.

The terms in the core vocabulary were defined and manually mapped to the ML ontologies participating in this endeavor through several rounds of consultations and working sessions. In 2016, the MLS-CG published an online proposal for MLS on the community group portal, and welcomed comments and suggestions from the research community and wider [12].

In this paper, we present the results of three years of MLS-CG efforts in the standardization of the encoding of ML experiments. MLS aims to support a high level of interoperability among scientific experiments concerning machine learning to foster reproducible research. MLS enables recording of machine learning studies and results as linked open data. MLS is beneficial to ML experiments Ecosystems (e.g., OpenML and *Research Objects* [13]) and ML Metadata Repositories (e.g., *WASOTA* [14]) by providing a more representative standard for their architectures. In OpenML, MLS is used to export all machine learning datasets, tasks, workflows, and runs as linked open data. This allows scientists to connect the results of their machine learning experiments to other knowledge sources, or to build novel knowledge bases for machine learning research.

This paper is organized as follows. In Section 2, we introduce and discuss the MLS core vocabulary and properties. Next, in Section 3, we present a discussion of the alignment of MLS with related ontologies. Furthermore, in Section 4 we discuss several use-cases of our proposed schema. Finally, in Section 5 we present a summary of contributions and avenues for future work.

[1] URL:https://www.openml.org/

[2] See www.w3.org/community/ml-schema/

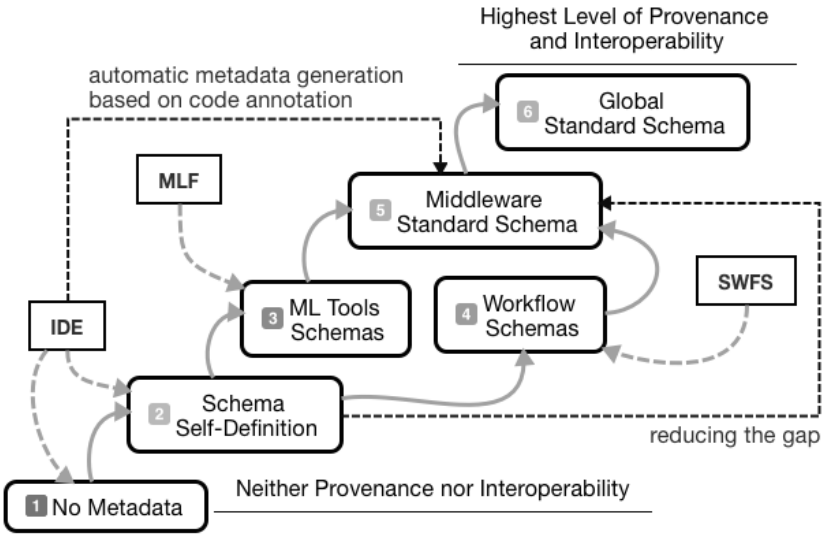


Fig. 1. Vertical and Horizontal Interoperability across ML Environments.

## 2. The ML-Schema

In this section, we introduce the MLS w.r.t. its aims and design principles. Next, we describe the MLS core vocabulary. Finally, we describe the properties defined within the MLS namespace.

### *2.1. The MLS aims and design principles*

The main aim of MLS is to provide a high-level standard to represent ML experiments in a concise, unambiguous and computationally processable manner. In particular, it aims to align existing ML ontologies and to support the development of more specific ontologies for particular purposes and applications.

To serve its purposes, MLS has to be compact but sufficiently comprehensive and easily extendable. To achieve such an aim, we chose to design MLS as a light-weight ontology that can be used as a basis for ontology development projects, markup languages, and data exchange standards. We then show how the MLS is open for further extensions and mappings to other resources.

For example, MLS can support *vertical* and *horizontal interoperability* across various ML environments [15]. Different ML platforms have different underlying schemes for representing data and metadata (see Figure 1: items 3 and 4: vertical interoperability). In turn, each schema may have a different level of engineering design, although representing the same information, i.e., two or more properties representing the same concept, but named differently (vertical level). In the worst-case scenario, a self-defined schema is used, which may lack clarity and may not follow any standard. In the best-case scenario, a generic format is defined and accepted by the academic community, serving as an upper-level ontology which is designed taking into account state-of-the-art (SOTA) ML-ontologies (see Figure 1: item 5). The idealization of this format would enable the interchange of machine learning metadata across different frameworks. Esteves et al [15] provides a more in-depth analysis of this problem.

It is worth noting that we do not propose yet another ontology for representing ML metadata, but rather work towards a global representation of ML metadata through existing SOTA ML ontologies. New terms in our representation suppress possible missing properties and/or concepts from SOTA ML Ontologies. We thus propose ML-Schema to act as a central point to connect existing and new ML schemata. We claim the gap can be further (significantly) reduced by achieving interoperability among SOTA schemata of those resources (see Figure 1: item 5) i.e. achieving the horizontal interoperability (Figure 1: item 6). Therefore, different groups of researchers could exchange SOTA metadata files in a transparent manner, e.g.: from OntoDM and MEX (`MLS.Schemadata=MLS.convert('myfile.ttl',MLS.Ontology.OntoDM, MLS.Ontology.MEX)`).

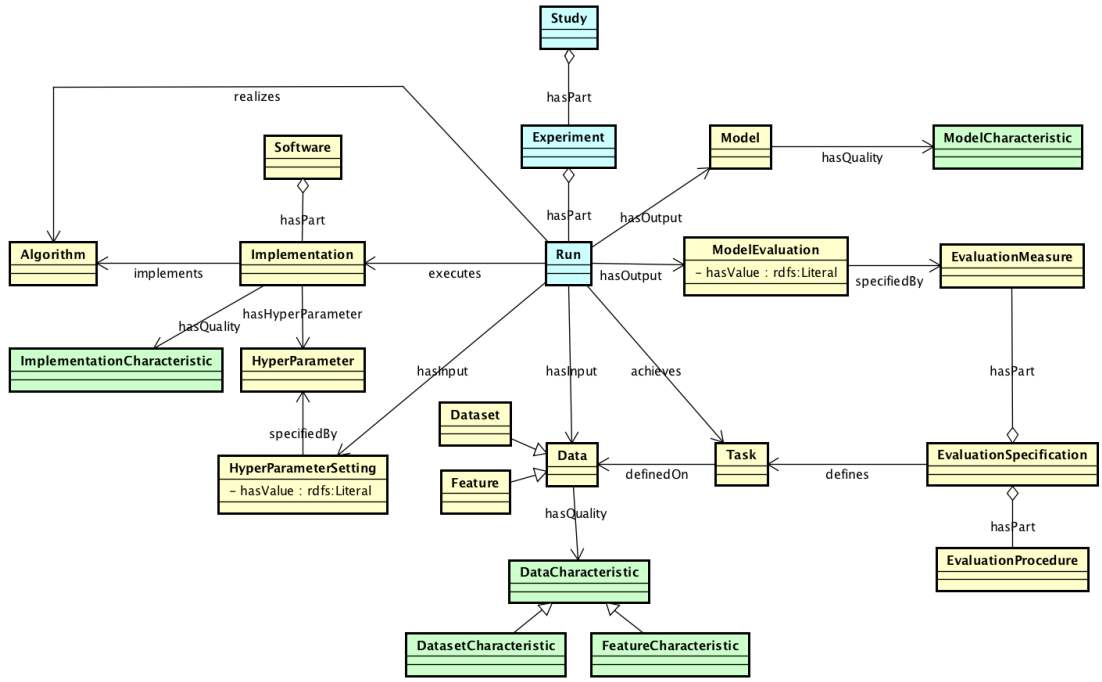


Fig. 2. The ML Schema core vocabulary. The diagram depicts Information Entities as yellow boxes, Processes as blue boxes, and Qualities as green boxes.

### 2.2. *MLS upper-level categories*

In this section, we present an overview of MLS upper-level categories. The diagram depicted in Figure 2 shows the complete MLS vocabulary. In general, the MLS vocabulary contains representations of three categories of entities we observe in the domain of machine learning experimentation: *information entities*, *process entities* and *quality entities*.

Information content entity is defined in the IAO[3] (Information Artefact Ontology) as "a generically dependent continuant that is about some thing." Examples of information entities in our vocabulary include: *task*, *data*, *dataset*, *feature algorithm*, *implementation*, *software*, *hyper-parameter*, *hyper-parameter setting*, *model*, *model evaluation*, *evaluation measure*, *evaluation specification* and *evaluation procedure*.

Process is defined in the BFO[4] (Basic Formal Ontology) [16] as a "an occurrent that has temporal proper parts and for some time t, p s-depends_on some material entity at t." In our vocabulary we deal mostly with planned processes, defined in OBI (Ontology of Biomedical Investigations) [17] as "a processual entity that realizes a plan which is the concretization of a plan specification." Examples of process entities in our vocabulary include: *run*, *experiment* and *study*. Processual entities (e.g., *run*) can have participants, which can be input (e.g., *data*) or output participants (e.g., *model*, *model evaluation*). Planned processes (e.g., *run*) can also execute a plan (e.g., in our case a plan is encoded in the algorithm *implementation*) and can achieve a planned objective represented as a *task* specification. Finally, our process entities form a parthood taxonomy: one *study* can have as parts *experiments*; and one *experiment* can have as parts *runs*.

Quality is defined in BFO as "a specifically dependent continuant that, in contrast to roles and dispositions, does not require any further process in order to be realized." PATO[5] (Phenotypic Quality Ontology) defines quality as a "a dependent entity that inheres in a bearer by virtue of how the bearer is related to other entities." Examples of quality entities in our domain include: *data characteristic*, *dataset characteristic*, *feature characteristic*, *model characteristic*, and *implementation characteristic*.

[3]URL: https://github.com/information-artifact-ontology/IAO/
[4]URL: http://purl.obolibrary.org/obo/bfo.owl
[5]URL: http://purl.obolibrary.org/obo/pato.owl

### 2.3. MLS core vocabulary

In the continuation of this section, we briefly discuss the core classes of MLS in more detail.

**Task** in MLS represents a formal specification of a process objective that needs to be completed or achieved (e.g.based on specific inputs and outputs). In general, a task is any piece of work that needs to be addressed in a data mining process. Example of possible sub-classes of tasks include: *classification*, *regression*, *clustering*, *feature selection*, *missing value imputation* and others. Examples of individuals include for example the task of *classification on Iris dataset*[6].

**Algorithm** in MLS, represents an algorithm specification described in a report, a scientific paper or just written on some media in a form of a pseudo code. This allows the potential users to extend the representation by adding algorithm provenance information, such as title, creator and others, by using for example the Dublin Core (DC) vocabulary[7]. Examples of possible sub-classes of algorithm include: *classification algorithm*, *regression algorithm*, *multi-label classification algorithm*. Examples of individuals include: *ID3 algorithm* [18] and *C4.5 algorithm* [19] as instances of the *classification algorithm*, *linear regression algorithm* as an instance of *regression algorithm* and others.

**Implementation** in MLS, represents an executable software implementation of a machine learning algorithm, script, or workflow. It is versioned, and sometimes belongs to a library (e.g. WEKA [20], RapidMiner[8]). This is represented by the part hood relation with **Software**. Implementations have *hyper parameters*. Potential users can extend the representation of implementations to include provenance information by using external vocabularies and ontologies. Example sub-classes of implementation can include: *learner implementation*, *data processing implementation*, *evaluation procedure implementation*, and others. Example of individuals include: *weka.J48*, *rapidminer.RandomForest*, *weka.evaluation.CrossValidation*, and others.

**HyperParameter** in MLS represents a prior parameter of an implementation, i.e., a parameter which is set before its execution (e.g. C, the complexity parameter, in weka.SMO implementation). Hyper-parameters are built in the implementation by design and they influence the implementation execution, when realized in a run process. Example of individuals of this class can include: *weka.SMO_C* (the C parameter in the WEKA's implementation of the support vector machines [21]), *weka.J48_M* (the M parameter in the WEKA's implementation of the C4.5 algorithm - named J48 in the implementation), *rapidminer.RandomForest_number_of_trees* (the number of trees parameter in the Rapidminer implementation of the Random forest algorithm [22]).

**HyperparameterSetting** class is used for representation of the parameter settings of the implementation that is realized in each specific run. This is done by using *hasValue* data property.

**Data** in MLS represent data items. In IAO, data item is defined as "information content entity that is intended to be a truthful statement about something (modulo, e.g., measurement precision or other systematic errors) and is constructed/acquired by a method which reliably tends to produce (approximately) truthful statements." Data used in machine learning experiments can appear in various levels of granularity and complexity depending on the task at hand. With regard to granularity, it can be a complete dataset (for instance, one main table and possibly other tables), a single table, a single feature (e.g., a column of a table), or only an instance (e.g., a row of a table), or a single feature-value pair. With regards to complexity, data items are characterized by their datatype, which may be arbitrarily complex (e.g., instead of a table it can be a graph). Finally, depending on the use case at hand, data descriptions can be extended by incorporating provenance information (e.g., name, description, URL, identifier, creator, publisher, and others) by reusing external vocabularies, such as DC, schema.org[9], and DCAT (Data Catalog Vocabulary)[10].

**Dataset and Feature** are represented as sub-classes of data. Example of individuals of the datasets class can include: *Iris dataset*, *FaceScrub dataset*, *IMDB-WIKI dataset*, and others. Examples of in-

[6]URL:https://archive.ics.uci.edu/ml/datasets/iris

[7]URL: https://www.dublincore.org/specifications/dublin-core/dces/

[8]URL:https://rapidminer.com/

[9]URL: https://schema.org/

[10]URL:https://www.w3.org/TR/vocab-dcat/

dividuals of features that appear in the *Iris dataset* are as follows: *sepal length*, *sepal width*, *petal length*, *petal width*, and *class*.

**DataCharacteristic** in MLS is used for representation of different data properties. This class has two sub-classes used for representation of properties of datasets (**DatasetCharacteristic**) and features (**FeatureCharacteristic**). Examples of such properties include: number of features, number of labels, number of instances, and others.

**Run** in MLS is an execution of an implementation of an algorithm on a machine (e.g., computer). Runs receive data and hyper-parameter settings at input. Runs are limited in time, have a start and an end point, and can be successful or failed. If successful, runs produce a specific result, such as a model and evaluations of model's performance. Although runs are called very differently in the different existing ontologies, the semantics are the same. Examples of individuals of run include: *process running SVMlib on Iris on Machine M on timestamp* t. Finally, depending on the use case at hand, runs can be extended by incorporating provenance information about a run (e.g., name, description, identifier, creator, and others) by reusing external vocabularies.

**Model** is defined as an generalization of the input data produced by an execution of an algorithm implementation in a specific run. Models have a dual nature: they can be treated as data structures and as such represented, stored and manipulated; on the other hand, they act as functions and are executed (e.g., in the case of predictive models they take as input unlabeled data examples and giving as output the prediction of target features). Models can also be divided into global or local ones. A global model has global coverage of a data set, i.e., it generalizes the whole data set. A local model, such as a pattern set, is a set of local hypotheses, i.e. each applies to a limited region of the data set. Example sub-classes of model class include: *decision tree*, *rule set*, *clustering*, *pattern set*, *bayesian network*, *neural network* and others. Example of an individual is *a decision tree built on Iris dataset using weka.ID3 implementation with default parameters*.

**ModelCharacteristic** are used for characterizing different properties of models. For example, if we have decision trees as the type of model, we can characterize the model with several propertied: *tree size*, *tree depth*, *number of leaves*, *number of internal nodes*, and others. The model properties are directly dependent on the model type (e.g., neural networks have different set of characteristics then decision trees).

**EvaluationProcedure** in MLS is used to represent different procedures to evaluate machine learning models. The evaluation procedure is dependent on the task at hand (e.g., different evaluation procedures are used for predictive modeling and for clustering). Examples of evaluation procedures include: *cross-validation*, *train-test validation*, *leave-one-out validation* and others.

**Evaluation measure** in MLS uniquely defines how to assess the performance of a model. For example, for the case of classification model we can assess the performance of the built model with a set of measures that include: *accuracy*, *precision*, *recall*, *F-measure*, and others. after it has been trained in a specific run. These measures are directly dependent on the task at hand (e.g., the evaluation measures for classification are different from evaluation measures for regression). There also exist evaluation measures that are task independent, such as *duration of the model training*, *duration of the model testing* and *consumption of memory*.

**EvaluationSpecification** in MLS represents a specification of a specific evaluation defined on a task by using specific evaluation procedure and evaluation measure. For example, we can represent a specification of an evaluation that is considering cross-validation in a classification setting and accuracy as an evaluation measure.

**Experiment** in MLS is a collection of runs. It can be used to group the runs in logically defined units that are used to address a specific research question or hypothesis.

**Study** in MLS has the highest level of granularity in representing collections of experiments. Studies are often the most natural product of a scientific investigation (that usually tackles several research questions or tests several hypothesis) and can be directly linked to certain claims and other products, such as research papers.

### *2.4. MLS Ontology properties*

Finally, in the following we list and briefly describe the properties modeled in MLS:

**achieves** A relation between a run and a task, where the run achieves specifications formulated by the task.
**definedOn** A relation between a task and either the data or an evaluation specification pertinent to this task.
**defines** The inverse relation of **definedOn**.
**executes** A relation between a run and an implementation that is being executed during the run.
**hasHyperParameters** A relation between implementation of a machine learning algorithm and its hyperparameter.
**hasInput** A relation between a run and data that is taken as input to the run.
**hasOutput** A relation between a run and either a model or model evaluation that is produced on its output.
**hasPart** A relation which represents a part-whole relationship holding between an entity and its part.
**hasQuality** A relation between entities and their various characteristics.
**implements** A relation between an information entity and a specification that it conforms to.
**realizes** A relation between a run and an algorithm, where the run realizes specifications formulated by the algorithm.
**specifiedBy** A relation between an entity and the information content entity that specifies it.

## 3. Alignment of MLS with related ontologies

In this section, we first give a brief overview of the related ontologies and vocabularies in the domain of machine learning and data mining. Second, we focus on the alignment of the proposed MLS with the related ontologies. Here, we perform an analysis of the different representations and discuss the alignments with the MLS core vocabulary.

### *3.1. Related ontologies*

The development of MLS was highly influenced by, initially independent, research of several groups on modeling the machine learning/data mining domain. Due to this the classes and relations presented in MLS re-appear in the current ML/DM ontologies and vocabularies. The following related ML ontologies are those that MLS is aligned to the moment. These include the OntoDM ontology [6], the DMOP ontology [7], the Exposé ontology [8] and the MEX vocabulary [11]. These alignments will be further described in the remainder of the section.

**The OntoDM ontology.** The Onto-DM ontology was initially designed to provide generic representations of principle entities in the area of data mining. In one of the preliminary versions of the OntoDM ontology [23], the authors decided to align the proposed ontology with the Ontology of Biomedical Investigations (OBI) [24] and consequently with the Basic Formal Ontology (BFO) at the top level[11], in terms of top-level classes and the set of relations. That was beneficial for structuring the domain more elegantly and establishing the basic differentiation of information entities, implementation entities, and processual entities. In this context, the authors proposed a horizontal description structure that includes three layers: a specification layer, an implementation layer, and an application layer [6]. The specification layer, in general, contains information entities (example classes are data mining task and data mining algorithm). The implementation layer, in general, contains entities that are realized in a process, such as parameters and implementations of algorithms, as well as models that are the output of the execution of algorithms on specific datasets. The application layer contains processual classes, such as the execution of the data mining algorithm.

**The DMOP ontology.** The DMOP ontology has been developed with a primary use case in meta-mining, that is meta-learning extended to an analysis of full DM processes [7]. At the level of both single algorithms and more complex workflows, it follows a very similar modeling pattern as described in the MLS. To support meta-mining, DMOP contains a taxonomy of algorithms used in DM processes which are described in detail in terms of their underlying assumptions, cost functions, optimization strategies, generated models or pattern sets, and other properties. Such a "glass box" approach which makes explicit internal algorithm characteristics allows meta-learners using DMOP to generalize over algorithms and their properties, including those algorithms which were not used for training meta-learners.

**The Exposé ontology.** The main goal of the Exposé ontology is to describe (and reason about) machine learning experiments in a standardized fashion and

[11] URL: http://basic-formal-ontology.org/

support a collaborative approach to the analysis of learning algorithms [8]. It is built on top of OntoDM by reusing several general ML classes and DMOP by reusing classes related to internal algorithm mechanism, as well as other general ontologies for experimentation, such as EXPO [25]. Its conceptualization is currently used in OpenML [9], as a way to structure data (e.g. database design) and share data (APIs). MLS will be used to export all OpenML data as linked open data (in RDF). For the sake of simplicity and comprehension, we further refer to the Exposé ontology as the OpenML vocabulary, or simply OpenML.

**The MEX vocabulary.** MEX has been designed to reuse existing ontologies (i.e., PROV-O, DublinCore[12], and DOAP[13]) for representing basic machine learning information. The aim is not to describe a complete data-mining process, which can be modeled by more complex and semantically refined structures. Instead, MEX was designed to provide a simple and lightweight vocabulary for exchanging machine learning metadata to achieve a high level of interoperability as well as supporting data management for ML outcomes.

### *3.2. Alignment analysis*

MLS provides a model for expressing data mining and machine learning algorithms, datasets, and experiments. In Table 1, we present the mapping between the terms present in the MLS and the current ML/DM ontologies and vocabularies. This mapping highlights how MLS is compatible with prior ontologies and how resources currently described in other ontologies can be described uniformly using MLS, hence allowing us to link currently detached machine learning resources.

In the remainder of this section, we discuss the analysis of the mappings of MLS core vocabulary with the existing ML ontologies, presented in the previous section.

#### *3.2.1.* ***Task***

In MLS, the *Task* class represents a formal description of a process that needs to be completed or achieved. We directly align it with the following concepts from the related ontologies: *OntoDM: "Data Mining Task"*, *DMOP: DM-Task*, *OpenML:TaskType*. In the MEX vocabulary the closest concept is *mexcore:ExperimentConfiguration*. We briefly discuss the representation rationale in each of the related ontologies.

**OntoDM.** OntoDM defines a data mining task as an objective specification that specifies the objective that a data mining algorithm needs to achieve when executed on a dataset to produce as output a generalization. It is represented as a subclass of the IAO: objective specification class, where objective specification is a directive information entity that describes and intended process endpoint. The data mining task is directly dependent on the datatypes of the data examples on which the task is defined, and is included directly in the task representations. This allows us to represent tasks defined by arbitrarily complex datatypes. The definition of data mining algorithm and generalizations is strongly dependent on the task definition.

OntoDM contains a taxonomy of data mining tasks. At the first level, they differentiate between four major task classes: predictive modeling task, pattern discovery task, clustering task, and probability distribution estimation task. Predictive modeling task is worked out in more detail. A predictive modeling task is defined on a pair of datatypes (one describing the part of the data example on the descriptive side and the other describing the part of the data example on the target/output side), they differentiate between primitive output prediction tasks (that include among others the traditional ML tasks such as classification and regression) and structured output prediction tasks (that include, among other, tasks such as multi-label classification, multi-target prediction, hierarchical multi-label classification).

**DMOP.** In DMOP, a task is any piece of work that is undertaken or attempted. A DM-Task is any task that needs to be addressed in the data mining process. DMOP's DM-Task hierarchy models all the major task classes: CoreDM-Task, DataProcessingTask, HypothesisApplicationTask, HypothesisEvaluationTask, HypothesisProcessingTask, InductionTask, ModelingTask, DescriptiveModelingTask, PredictiveModelingTask, and PatternDiscoveryTask.

**OpenML.** OpenML differentiates a TaskType (e.g. classification, regression, clustering,...) and Taskinstances. The TaskType defines which types of inputs are given (e.g. a dataset, train-test splits, optimization measures) and which outputs are expected (e.g. a model, predic-

[12] http://dublincore.org
[13] http://usefulinc.com/doap/

Table 1
Full alignment between the core terms of ML-Schema and the related vocabularies.

| ML-Schema | OntoDM-core | DMOP | OpenML/Exposé | MEX Vocabulary |
|---|---|---|---|---|
| Task | Data mining task | DM-Task | Task | `mexcore:ExperimentConfiguration` |
| Algorithm | Data mining algorithm | DM-Algorithm | Algorithm | `mexalgo:Algorithm` |
| Software | Data mining software | DM-Software | Software | `mexalgo:Tool` |
| Implementation | Data mining algorithm implementation | DM-Operator Algorithm implementation | Flow/Implementation | `mexalgo:Implementation` |
| HyperParameter | Parameter | Parameter | Parameter | `mexalgo:HyperParameter` |
| HyperParameterSetting | Parameter setting | OpParameterSetting | Parameter setting | N/A |
| Study | Investigation | N/A | Study | `mexcore:Experiment` |
| Experiment | N/A | DM-Experiment | Experiment | N/A |
| Run | Data mining algorithm execution | DM-Operation | Algorithm execution | `mexcore:Execution` |
| Data | Data item | DM-Data | Data | `mexcore:Example` |
| Dataset | DM dataset | DataSet | Dataset | `mexcore:Dataset` |
| Feature | N/A | Feature | Feature | `mexcore:Feature` |
| DataCharacteristic | Data specification | DataCharacteristic | Dataset specification | N/A |
| DatasetCharacteristic | Dataset specification | DataSetCharacteristic | Data quality | N/A |
| FeatureCharacteristic | Feature specification | FeatureCharacteristic | Description | N/A |
| Model | Generalization | DM-Hypothesis (DM-Model / DM-PatternSet) | Model | `mexcore:Model` |
| ModelCharacteristic | Generalization quality | HypothesisCharacteristic | Model Structure, Parameter, ... | N/A |
| ModelEvaluation | Generalization evaluation | ModelPerformance | Evaluation | N/A |
| EvaluationMeasure | Evaluation datum | ModelEvaluationMeasure | Evaluation measure | `mexperf:PerformanceMeasure` |
| EvaluationProcedure | Evaluation algorithm | ModelEvaluationAlgorithm | Performance Estimation | N/A |

tions,...). On the other hand, a Task contains specific dataset, splits, etc. It can be seen as an individual of the class.

**MEX.** MEX has a higher level of abstraction, designed for representing ML executions and related metadata and not DM tasks. There are specific classes for representing specific ML standards. This information could be obtained from Learning Problem + Learning Method + Algorithm Class in a more concise level.

- Learning Problem: Association, Classification, Clustering, Metaheuristic, Regression, Summarization, ...
- Learning Method: Supervised Learning, Unsupervised Learning, Semi-supervised Learning, Reinforcement Learning, ...
- Algorithm Class: ANN, ILP, Bagging, Bayes Theory, Boosting, Clustering, Decision Trees, Genetic Algorithms, Logical Representations, Regression Functions, Rules, Support Vector Networks, ...

As an `:ExperimentConfiguration` may have many `:Executions` and an `:Experiment` may have many `:ExperimentConfigurations`, these can be aligned to a `mls:Task`.

### 3.2.2. *Algorithm*

In MLS, the *Algorithm* class represents an algorithm regardless of its software implementation. We directly align it with the following concepts from the related ontologies: *OntoDM:"Data Mining Algorithm"*, *DMOP: DM-Algorithm*, *OpenML: Algorithm*, and *MEX: mexalgo:Algorithm*. We briefly discuss the representation rationale in each of the related ontologies.

**OntoDM** In OntoDM, authors differentiate between three aspects of algorithms: algorithm as a specifica-

tion, algorithm as an implementation, and the process of executing an algorithm. Data mining algorithm (as a specification) is represented as a subclass of IAO: algorithm. In this sense, a data mining algorithm is defined as an algorithm that solves a data mining task and as a result outputs a generalization and is usually published/described in some document (journal/conference/workshop publication or a technical report).

In OntoDM, it is given a higher-level taxonomy of algorithms. At the first level, it is differentiated between single generalization algorithms (algorithms that produces a single generalization as a result) and ensemble algorithms (algorithms that produce an ensemble of generalizations as a result). At the second level, the taxonomy follows the taxonomy of tasks. This modular and generic approach allows easy extensions to characterize each algorithm class with its own distinctive set of characteristics that can be represented as qualities.

**DMOP.** A DM-Algorithm is a well-defined sequence of steps that specifies how to solve a problem or perform a task. It typically specifies an input and an output. A DM-Algorithm is an algorithm that has been designed to perform any of the DM tasks, such as feature selection, missing value imputation, modeling, and induction. The higher-level classes of the DM-Algorithm hierarchy correspond to DM-Task types. Immediately below are broad algorithm families or what data miners more commonly call paradigms or approaches. The Algorithm hierarchy bottoms out in individual algorithms such as CART, Lasso or ReliefF. A particular case of a DM-Algorithm is a Modeling (or Learning) algorithm, which is a well-defined procedure that specifies data as input and an output in the form of models or patterns.

**OpenML.** OpenML currently does not abstract over algorithms anymore, it simply has 'implementations'. The underlying reasoning is that algorithms can come in endless variations, including hybrids that combine multiple pre-existing algorithms. Classifying every implementation as a specific type of algorithm is therefore not trivial and hard to maintain. Instead, to organize implementations, OpenML has 'tags', so that anybody can tag algorithms with certain keywords, including the type of algorithm that is implemented. Hence, a hybrid algorithm can have multiple tags.

**MEX.** Sharing the solution by OpenML, MEX labels different levels of ML algorithms in Algorithm class instead of specific algorithm characterisations. As much as more precise information is needed, related classes could be instantiated, such as Learning Problem + Learning Method + Algorithm Class + Implementation.

### *3.3.* ***Implementation***

In MLS, the *Implementation* class represents an executable implementation of a machine learning algorithm, script, or workflow. It is versioned, and sometimes belongs to a library (e.g. WEKA). We directly align it with the following concepts from the related ontologies: *OntoDM: "Data mining algorithm implementation"*, *DMOP: DM-Operator / DM-Workflow*, *OpenML: Flow / Implementation* and *MEX: mexalgo:Implementation*. We briefly discuss the representation rationale in each of the related ontologies.

**OntoDM.** In OntoDM, a data mining algorithm execution is a subclass of `SWO:information` processing, which is an `OBI:planned` process. Planned processes realize a plan which is a concretization of a plan specification. A data mining algorithm execution realizes (executes) a data mining operator, has as input a dataset, has as output a generalization, has as agent a computer, and achieves as a planned objective a data mining task.

Data mining operator is a role of a data mining algorithm implementation that is realized (executed) by a data mining algorithm execution process. The data mining operator has information about the specific parameter setting of the algorithm, in the context of the realization of the operator in the process of execution. The parameter setting is an information entity which is a quality specification of a parameter.

**OpenML.** OpenML does not distinguish between 'operators' and 'workflows', because the line is often very blurry. Many algorithms have complex internal workflows to preprocess the input data and make them more robust. Also, many environments (e.g. R, Matlab, etc.) do not have the concept of operator; they just have function calls, which are part of scripts. Hence, in OpenML, every implementation is called a Flow, which can be either atomic or composite.

**DMOP and MEX.** The Implementation class of MLS is aligned to DMOP class DM-Operator: a programmed, executable implementation of a DM-Algorithm. Implementation in MEX is meant to represent the *Software* Implementation and has no link to

the algorithm itself. Examples are Weka, SPSS, Octave, DL-Learner.

### 3.4. *HyperParameter*

The MLS HyperParameter class represents a prior parameter of an implementation, i.e., a parameter which is set before its execution (e.g. C, the complexity parameter, in `weka.SMO`). We directly align it with the following concepts from the related ontologies: *OntoDM: Parameter*, *OpenML: Parameter*, *MEX:AlgorithmParameter* and *mexalgo:HyperParameter* (term under a proposal status), and *DMOP's OperatorParameter*.

In OntoDM, authors represent a data mining algorithm implementation as a subclass of OBI: plan is a concretization of a data mining algorithm. Data mining algorithms have as qualities parameters that are described by a parameter specification. A parameter is a quality of an algorithm implementation, and it refers the data provided as input to the algorithm implementation that influences the flow of the execution of algorithm realized by a data mining operator that has information about the specific parameter setting used in the execution process.

### 3.5. *Data*

In MLS, the Data class represents a data item composed of data examples and it may be of a various level of granularity and complexity. We directly align it with the following concepts from the related ontologies: *OntoDM: data item*, *OpenML: Data*, *DMOP: DM-Data* and *MEX mexcore:Example*. Furthermore, we align the *Dataset* class with the following concepts: *OntoDM: DM-dataset*, *DMOP: DataSet* and *MEX mexcore:Dataset* (as metadata). We briefly discuss the representation rationale in the related ontologies.

**OntoDM.** OntoDM imports the IAO class dataset (defined as 'a data item that is an aggregate of other data items of the same type that have something in common') and extends it by further specifying that a DM dataset has part data examples. OntoDM-core also defines the class dataset specification to enable characterization of different dataset classes. It specifies the type of the dataset based on the type of data it contains. In OntoDM, the authors model the data characteristics with a data specification entity that describes the datatype of the underlying data examples. For this purpose, we import the mechanism for representing arbitrarily complex datatypes from the OntoDT ontology. Using data specifications and the taxonomy of datatypes from the OntoDT ontology, in OntoDM-core have a taxonomy of datasets.

**DMOP.** In SUMO, Data is defined as an item of factual information derived from measurement or research. In IAO, Data is an alternative term for 'data item': 'an information content entity that is intended to be a truthful statement about something (modulo, e.g., measurement precision or other systematic errors) and is constructed/acquired by a method which reliably tends to produce (approximately) truthful statements'. In the context of DMOP, DM-Data is the generic term that encloses different levels of granularity: data can be a whole dataset (one main table and possibly other tables), or only a table, or only a feature (column of a table), or only an instance (row of a table), or even a single feature-value pair.

**MEX.** In MEX, it is possible to represent even each instance (mexcore:Example) and each feature (mexcore:Feature) of the dataset.

### 3.6. *Model*

We define *Model* as a generalization of a set of training data able to predict values for unseen instances. It is an output from the execution of a data mining algorithm implementation. Models have a dual nature: they can be treated as data structures and as such represented, stored and manipulated; on the other hand, they act as functions and are executed, taking as input data examples and giving as output the result of applying the function to a data example. Models can also be divided into global or local ones. We directly align it with the following concepts from the related ontologies: *OntoDM: Generalization*, *OpenML: Model*; *DMOP: DM-Hypothesis* (with main subclasses: *DM-Model*, *DM-PatternSet*). We briefly discuss the representation rationale in the related ontologies.

**OntoDM.** In OntoDM, authors take generalization to denote the outcome of a data mining task. They consider and model three different aspects of generalizations: the specification of a generalization, a generalization as a realizable entity, and the process of executing a generalization. Generalizations have a dual nature. They can be treated as data structures and as such represented, stored and manipulated. On the other hand, they act as functions and are executed, taking as

input data examples and giving as output the result of applying the function to a data example. In OntoDM, a generalization is defined as a sub-class of the BFO class realizable entity. It is an output from a data mining algorithm execution.

The dual nature of generalizations in OntoDM is represented with two classes that belong to two different description layers: generalization representation, which is a sub-class of information content entity and belongs to the specification layer, and generalization execution, which is a subclass of planned process and belongs to the application layer.

**DMOP.** By Hypothesis, DMOP actually meant roughly ML models. They introduced the concept of a 'hypothesis' to differentiate ML models from pattern sets. On the other hand, the DM-PatternSet represents a pattern set, as opposed to a model which by definition has global coverage, is a set of local hypotheses, i.e. each applies to a limited region of the sample space.

### *3.7.* ***Run***

An MLS run is an execution of an implementation on a machine (computer). If successful, it often has a specific result, such as a model and evaluations of that model's performance. Although runs are called very differently in the different existing ontologies, the semantics are the same. We directly align it with the following concepts from the related ontologies: *OpenML: Run*; *DMOP: DM-Process* (i.e., execution), *OntoDM: Data mining algorithm execution*, and *mexcore:Execution* (singly *mexcore:SingleExecution*,*mexcore:OverallExecution*).

### *3.8.* ***EvaluationMeasure***

An MLS evaluation measure unique defines how to evaluate the performance of a model after it has been trained in a specific run. We directly align it with the following concepts from the related ontologies: *OpenML: EvaluationMeasure*, *DMOP: Measure*, *OntoDM:Evaluation datum* and *mexperf:PerformanceMeasure*. In DMOP, however, there exist subclasses, such as ComputationalComplexityMeasure, HypothesisEvaluationMeasure, and ModelComplexityMeasure.

### *3.9.* ***Study***

An MLS study is a collection of runs that belong together to perform some kind of analysis on its results. This analysis can be general or very specific (e.g. an hypothesis test). It can also be linked to files, data, that belong to it. Studies are often the most natural product of a scientific investigation, and can be directly linked to certain claims and other products, such as research papers. As shown in Table 1 existing ontologies call this either a study, investigation or experiment, although the semantics are similar.

## 4. Use cases

To elucidate the benefits of MLS, in this section we present four use cases where MLS can be utilized to foster the reproducibility of experiments. In particular, we show how previous research can benefit from the existence of an upper ontology which interlinks several vocabularies used for the exchange of experiment data and metadata. The first use case shows how MLS can be used to represent machine learning studies on an application domain. The second use case discusses the potential MSL has to be used together with the Open Provenance Model for Workflows and Research Objects. The third use case shows the use of MLS in the OpenML platform. Finally, in the fourth use case, we discuss the potential of using MLS in the future to support deep learning models.

### *4.1. Representing Machine Learning Studies*

In this use case, we illustrate how MLS can be used for representing machine learning studies. Exposing metadata about performed study may be of use for possible collaborators who may wish to analyse research networks and try to assess the 'trustwothiness' of what is published in the literature. Such information that a study is done within a funded project, may increase their level of trust to the published results.

In Figure 3, we show an example describing ML study (`:study1`) and the corresponding dataset (`:mtl_dataset`), providing reference to a publication (`:article1`), and acknowledging a funding body (`:EPSRC`). In addition, we provide the RDF code listing in Figure 4. This example refers to the article "Multi-Task Learning with a Natural Metric for Quantitative Structure Activity Relationship Learning" by Sadawi et al which re-

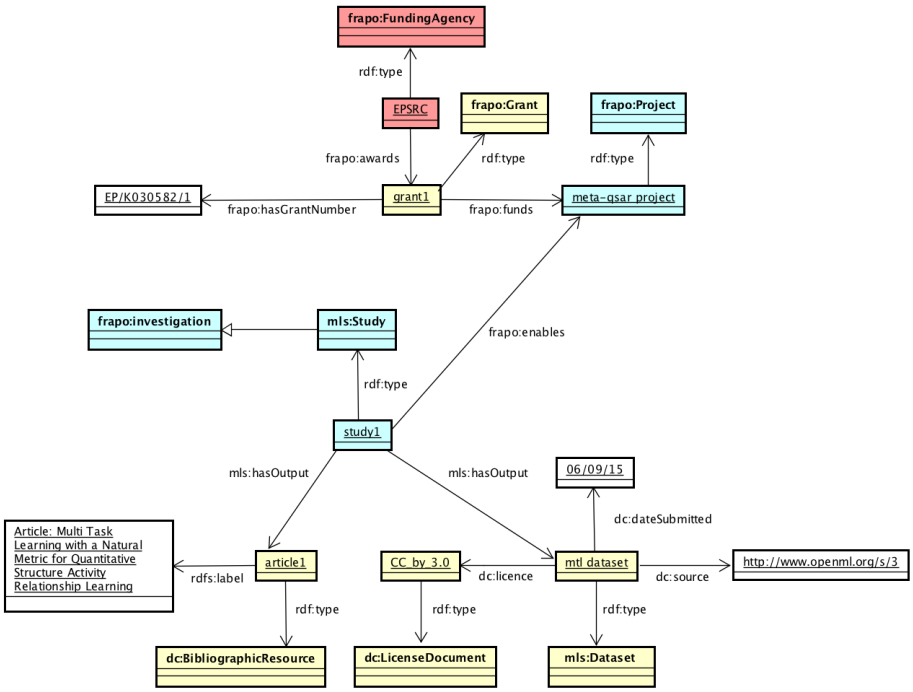


Fig. 3. An example of representation of a machine learning study using MLS and FRAPO.

ports on the ML study carried out within the Meta-QSAR project[14] (`:meta-qsar_project`) funded by `:EPSRC` (`:grant1` with number EP/K030582/1). The referred dataset is freely available in OpenML[15]. For this use case we are linking MLS with the Funding, Research Administration and Projects Ontology (FRAPO)[16].

### 4.2. *Open Provenance Model for Workflows and Research Objects*

It is often crucial to know exactly which data was used to train a machine learning model, where this data came from, and how it was processed before modelling. MLS is compatible with the *Open Provenance Model for Workflows (OPMW)* [26] and *Research Objects* [13]. This allows machine learning experiments to be described in a uniform way that preserves the provenance of data and models.

The term *provenance*, in computer science and scientific research, means metadata about the origin, derivation or history of data or thing. For instance, in biology or chemistry, we track steps of experimental processes to enable their reproduction. In computer science, we track the creation, editing and publication of data, including their reuse in further processes. The PROV data model for provenance was created, founded on previous efforts such as Open Provenance Model (OPM) [27], and later became recommended by W3C [28]. The PROV Ontology (PROV-O), also recommended by W3C [29], expresses the PROV Data Model using the OWL language. PROV-O provides a set of classes, properties, and restrictions that can be used to represent and exchange provenance information generated in various systems. The Open Provenance Model for Workflows (OPMW) is an ontology for describing workflow traces and their templates which extends PROV-O and the ontology P-plan designed to represent plans that guided the execution of processes [26]. Figure 5 presents the mapping of the MLS directly to OPMW and indirectly to PROV-O and P-plan.

Belhajjame et al. [13] proposed a suite of ontologies for preserving workflow-centric *Research Objects*. The ontologies use and extend existing widely used ontologies, including PROV-O. Especially, the two ontologies from the suite, the Workflow Description Ontology (wfdesc), used to describe the workflow

[14]URL: http://www.meta-qsar.org/index.html
[15]URL:https://www.openml.org/s/3
[16]URL: http://purl.org/cerif/frapo

```
@prefix : <http://example.org#> .
@prefix mls: <http://www.w3.org/ns/mls#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
@prefix dc:  <http://purl.org/dc/elements/1.1/>> .
@prefix frapo: <http://purl.org/cerif/frapo/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .


dc:BibliographicResource rdf:type owl:Class .


mls:Study rdfs:subClassOf frapo:Investigation .

:EPSRC rdf:type frapo:FundingAgency ,
                owl:NamedIndividual ;

       frapo:awards :grant1 .

:article1 rdf:type dc:BibliographicResource ,
                   owl:NamedIndividual ;

          rdfs:label "Article: Multi Task Learning with
          a Natural Metric for Quantitative Structure Activity
          Relationship Learning" .

:grant1 rdf:type frapo:Grant ,
                 owl:NamedIndividual ;

        frapo:hasGrantNumber "EP/K030582/1" ;

        frapo:funds :meta-qsar_project .


:meta-qsar_project rdf:type owl:NamedIndividual ,
                            foaf:Project .

:mtl_dataset rdf:type owl:NamedIndividual ,
                      mls:Dataset ;

             dc:licence :CC_by_3.0 ;
             dc:dateSubmitted "06/09/15" ;
             dc:source "http://www.openml.org/s/3" .

:CC_by_3.0 rdf:type owl:NamedIndividual ,
                    dc:LicenceDocument .

:study1 rdf:type owl:NamedIndividual ,
                mls:Study ;

        frapo:enables :meta-qsar_project ;

        frapo:hasOutput :mtl_dataset .
```

Fig. 4. An example of a RDF code listing ilustrating the use case depicted in Figure 3.

specifications, and the Workflow Provenance Ontology (wfprov), used to describe the provenance traces obtained by executing workflows, follow a very similar conceptualization of workflows to that of OPMW and map to MLS.

### 4.3. OpenML

The OpenML platform contains thousands of machine learning experiments, with millions runs using thousands of machine learning workflows on thousands of datasets. However, in themselves, these experiments form another island of data disconnected to the rest of the world. To remedy this, we have used MLS to describe all of these experiments as linked open data, so that scientists can connect their machine learning experiments to other knowledge sources, or build novel knowledge bases for machine learning research.

This is achieved through an export function that reads in OpenML's current JSON descriptions of datasets, tasks, workflows, and runs, and emits an RDF description using the MLS schema. This functionality is available as an open source Java library[17]. OpenML also supports this export functionality on the platform itself. In the web interface (openml.org) every dataset, task, workflow (flow), and run page has an RDF export button that returns the RDF description of that object, linked to other objects by their OpenML IDs. This functionality is also available via predictable URLs in the format https://www.openml.org/{type}/{id}/rdf, where *type* is either *d* (dataset), *t* (task), *f* (flow), or *r* (run), and *id* the OpenML ID of that object. Hence, the RDF description of dataset 2 can be obtained via https://www.openml.org/d/2/rdf.

As such, OpenML data becomes part of the Semantic Web, which allows scientists to link it to other data and reuse it in innovate new ways.

In Figure 7, we illustrate an annotation using MLS of an example derived from the OpenML portal. This example describes entities involved to model a single run of the implementation of a logistic regression algorithm from a Weka machine learning environment. The referenced individuals can easily be looked up online. For instance, run 100241 can be found on http://www.openml.org/r/100241. In addition, in Figure 6 we present the RDF listing of the annotation.

### 4.4. Deep Learning

This use case can also be described as a possible future work of MLS, where it is extended to support Deep Learning (DL) models.

As an initiative of Microsoft and Facebook, a recently created community group called Open Neural Network Exchange (ONXX)[18] aims to allow users to share their Neural Network models and transfer them between frameworks. At the moment, it covers import/export to 3 different frameworks, while libraries for other 5 frameworks are under development or have partial support.

DL models have some requirements that MLS cannot describe at the moment – information such as the

[17]The library is available on https://github.com/ML-Schema/openml-rdf

[18]https://onnx.ai/

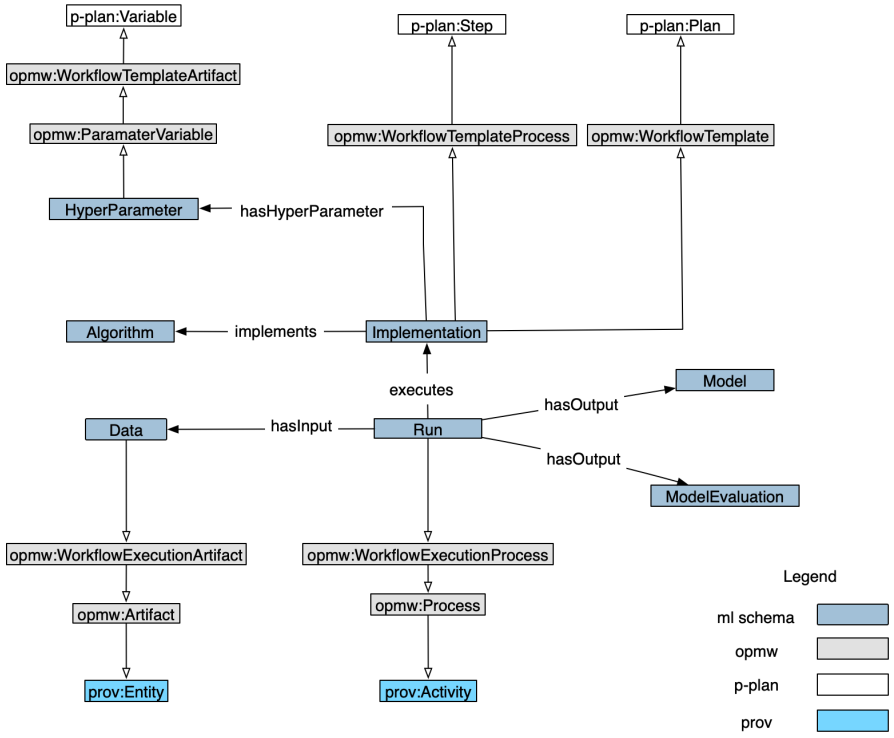


Fig. 5. The mapping of MLS to OPMW, PROV-O and P-plan.

number of layers and neurons, weights, and pre-trained models – as it only contains the HyperParameter class that is not able to store this additional information.

Unfortunately, the ONNX initiative does not provide an ontology; instead, their operators are described in the project GitHub documentation, while their terms are hardly defined in C code. On the other hand, the extension of the MLS ontology by adding new properties based on those terms would benefit not only the MLS, but all the aligned ontologies described in this work, that would instantly be able to use those properties to extend their models and support the description of DL models and experiments.

## 5. Conclusions and Future Work

In this paper we presented ML-Schema, a lightweight but sufficiently comprehensive and extendable ontology for the description of Machine Learning which supports the description and open publication of such experiments in an interchangeable format. We show the extension of its expressiveness and how the MLS ontology was designed to be aligned with several ML ontologies, such as DMOP, OntoDM, MEX, and Exposé. It was also possible to elucidate through use cases the capabilities of our work, such as the usage of MLS format for exporting ML experiments to RDF format in the OpenML framework, its extension of that provides direct support to the OPMW and indirect to the PROV-O ontology, as well as the possible extension to elucidate the description of DL experiments. Such extension will be handled in the future discussions of the MLS Community Group, that welcomes everyone interested in extending our format to achieve better support for description of ML experiments in an interchangeable format. Future works may also include another converters, such as for MyExperiment or many other e-Science platforms.

## Acknowledgements

Gustavo Correa Publio acknowledges the support of the Smart Data Web BMWi project (GA01MD15010B) and CNPq Foundation (201808/2015-3). Agnieszka Ławrynowicz acknowledges the support from the National Science Centre, Poland, within the grant number 2014/13/D/ST6/02076. Panče Panov acknowledges

the support of the Slovenian Research Agency within the grant J2-9230.

```
@prefix : <http://example.org#> .
@prefix mls: <http://www.w3.org/ns/mls#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .

:run100241 rdf:type owl:NamedIndividual ,
        mls:Run ;
        mls:executes  :wekaLogistic ;
        mls:hasInput  :credit-a ,
                      :wekaLogisticMSetting29 ,
                      :wekaLogisticRSetting29 ;
        mls:hasOutput :modelEvaluation100241 ,
                      :wekaLogisticModel100241 ;
        mls:realizes :logisticRegression ;
        mls:achieves :task29 .

:wekaLogistic rdf:type owl:NamedIndividual ,
        mls:Implementation ;
        mls:hasHyperParameter :wekaLogisticC ,
                    :wekaLogisticDoNotCheckCapabilities ,
                    :wekaLogisticM ,
                    :wekaLogisticOutputDebugInfo ,
                    :wekaLogisticR ;
        mls:implements :logisticRegression .

:weka rdf:type mls:Software,
    mls:hasPart :wekaLogistic.

:logisticRegression rdf:type owl:NamedIndividual ,
        mls:Algorithm .

:wekaLogisticC rdf:type owl:NamedIndividual ,
        mls:HyperParameter .
:wekaLogisticDoNotCheckCapabilities rdf:type owl:NamedIndividual ,
       mls:HyperParameter .
:wekaLogisticM rdf:type owl:NamedIndividual ,
        mls:HyperParameter .
:wekaLogisticOutputDebugInfo rdf:type owl:NamedIndividual ,
        mls:HyperParameter .
:wekaLogisticR rdf:type owl:NamedIndividual ,
        mls:HyperParameter .

:wekaLogisticMSetting29 rdf:type owl:NamedIndividual ,
        mls:HyperParameterSetting ;
        mls:specifiedBy :wekaLogisticM ;
        mls:hasValue -1 .

:wekaLogisticRSetting29 rdf:type owl:NamedIndividual ,
        mls:HyperParameterSetting ;
        mls:specifiedBy :wekaLogisticR ;
        mls:hasValue "1.0E-8"^^xsd:float .

:credit-a rdf:type owl:NamedIndividual ,
        mls:Dataset ;
        mls:hasQuality :defaultAccuracy ,
                       :numberOfFeatures ,
                       :numberOfInstances .

:defaultAccuracy rdf:type owl:NamedIndividual ,
        mls:DatasetCharacteristic ;
        mls:hasValue "0.56"^^xsd:float .

:numberOfFeatures rdf:type owl:NamedIndividual ,
        mls:DatasetCharacteristic ;
        mls:hasValue "16"^^xsd:long .

:numberOfInstances rdf:type owl:NamedIndividual ,
        mls:DatasetCharacteristic ;
        mls:hasValue "690"^^xsd:long .

:wekaLogisticModel100241 rdf:type owl:NamedIndividual ,
        mls:Model .

:modelEvaluation100241 rdf:type owl:NamedIndividual ,
        mls:ModelEvaluation ;
        mls:specifiedBy :predictiveAccuracy ;
        mls:hasValue 0.8478 .

:predictiveAccuracy rdf:type owl:NamedIndividual ,
        mls:EvaluationMeasure .

:task29 rdf:type owl:NamedIndividual ,
        mls:Task ;
        mls:definedOn :credit-a .

:evaluationSpecification1 rdf:type owl:NamedIndividual ,
        mls:EvaluationSpecification ;
        mls:defines :task29 ;
        mls:hasPart :TenFoldCrossValidation ,
                    :predictiveAccuracy .

:TenFoldCrossValidation rdf:type owl:NamedIndividual ,
        mls:EvaluationProcedure .
```

Fig. 6. An example instantiation of MLS derived from OpenML.

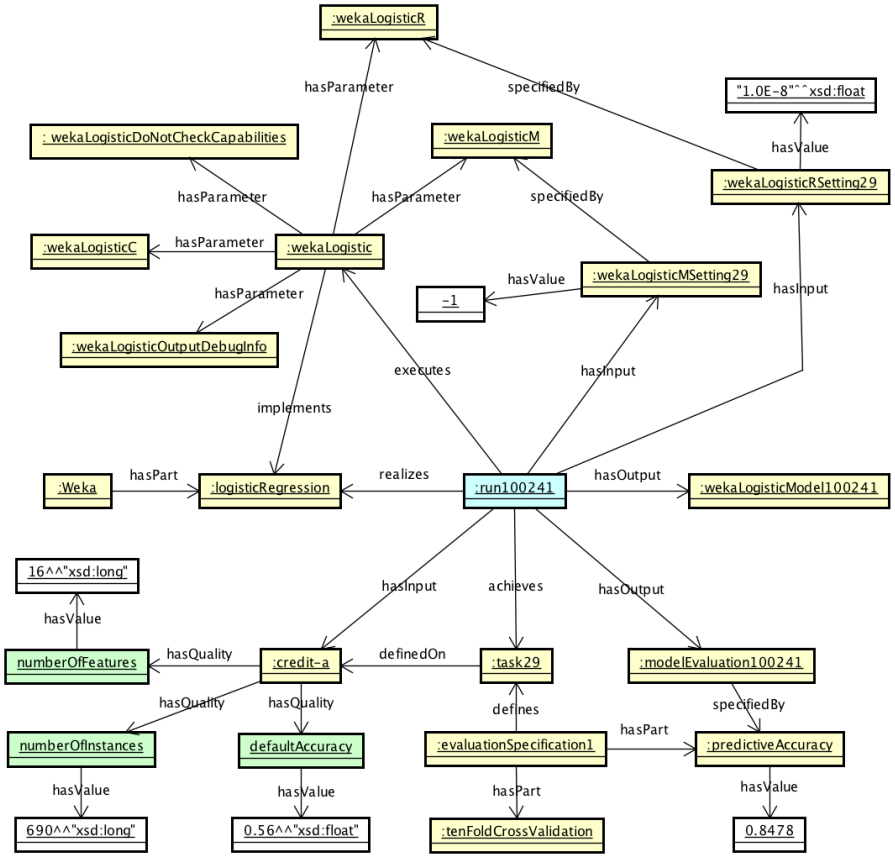


Fig. 7. An example illustrating a single run of an ML algorithm implementation. The diagram depicts Information Entities as yellow boxes, Processes as blue boxes, and Material Entities as red boxes..

## References


[1] Y. Gil, V. Ratnakar, J. Kim, P. Gonzalez-Calero, P. Groth, J. Moody and E. Deelman, Wings: Intelligent Workflow-Based Design of Computational Experiments, *IEEE Intelligent Systems* **26**(1) (2011), 62–72, ISSN 1541-1672. doi:10.1109/MIS.2010.9.

[2] O. Tcheremenskaia, R. Benigni, I. Nikolova, N. Jeliazkova, S.E. Escher, M. Batke, T. Baier, V. Poroikov, A. Lagunin, M. Rautenberg et al., OpenTox predictive toxicology framework: toxicological ontology and semantic media wiki-based OpenToxipedia, in: *Journal of biomedical semantics*, Vol. 3, BioMed Central, 2012, p. 7.

[3] D. De, R. Carole and G.R. Stevens, The design and realisation of the myexperiment virtual research environment for social sharing of workflows (2008).

[4] T.R. Gruber, A translation approach to portable ontology specifications, *KNOWLEDGE ACQUISITION* **5** (1993), 199–220.

[5] P. Panov, S. Džeroski and L. Soldatova, OntoDM: An ontology of data mining, in: *Data Mining Workshops, 2008. ICDMW'08. IEEE International Conference on*, IEEE, 2008, pp. 752–760.

[6] P. Panov, L. Soldatova and S. Džeroski, Ontology of core data mining entities, *Data Mining and Knowledge Discovery* **28**(5–6) (2014), 1222–1265.

[7] C.M. Keet, A. Lawrynowicz, C. d'Amato, A. Kalousis, P. Nguyen, R. Palma, R. Stevens and M. Hilario, The Data Mining OPtimization Ontology, *J. Web Sem.* **32** (2015), 43–53. doi:10.1016/j.websem.2015.01.001. https://doi.org/10.1016/j.websem.2015.01.001.

[8] J. Vanschoren and L. Soldatova, Exposé: An ontology for data mining experiments, in: *International workshop on third generation data mining: Towards service-oriented knowledge discovery (SoKD-2010)*, 2010, pp. 31–46.

[9] J. Vanschoren, J.N. Van Rijn, B. Bischl and L. Torgo, OpenML: networked science in machine learning, *ACM SIGKDD Explorations Newsletter* **15**(2) (2014), 49–60.

[10] J. Vanschoren, H. Blockeel, B. Pfahringer and G. Holmes, Experiment databases, *Machine Learning* **87**(2) (2012), 127–158, ISSN 1573-0565. doi:10.1007/s10994-011-5277-0. https://doi.org/10.1007/s10994-011-5277-0.

[11] 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: *Proceedings of the 11th International Conference on Semantic Systems*, ACM, 2015, pp. 169–176.

[12] D. Esteves, A. Lawrynowicz, P. Panov, L. Soldatova, T. Soru and J. Vanschoren, ML Schema Core Specification, draft report, W3C Machine Learning Schema Community Group, 2016, http://www.w3.org/2016/10/mls/.

[13] K. Belhajjame, J. Zhao, D. Garijo, M. Gamble, K.M. Hettne, R. Palma, E. Mina, Ó. Corcho, J.M. Gómez-Pérez, S. Bechhofer, G. Klyne and C.A. Goble, Using a suite of ontologies for preserving workflow-centric research objects, *J. Web Sem.* **32** (2015), 16–42. doi:10.1016/j.websem.2015.01.003. http://dx.doi.org/10.1016/j.websem.2015.01.003.

[14] C.B. Neto, D. Esteves, T. Soru, D. Moussallem, A. Valdestilhas and E. Marx, WASOTA: What Are the States Of The Art?, in: *SEMANTiCS (Posters, Demos, SuCCESS)*, 2016.

[15] D. Esteves, P. N. Mendes, D. Moussallem, J. Duarte, A. Zaveri, J. Lehmann, C. Neto and M.C. Cavalcanti, MEX Interfaces:Automating Machine Learning Metadata Generation, 2016. doi:10.1145/2993318.2993320.

[16] R. Arp, B. Smith and A.D. Spear, *Building Ontologies with Basic Formal Ontology*, The MIT Press, 2015. ISBN 0262527812, 9780262527811.

[17] A. Bandrowski, R. Brinkman, M. Brochhausen, M.H. Brush, B. Bug, M.C. Chibucos, K. Clancy, M. Courtot, D. Derom, M. Dumontier, L. Fan, J. Fostel, G. Fragoso, F. Gibson, A. Gonzalez-Beltran, M.A. Haendel, Y. He, M. Heiskanen, T. Hernandez-Boussard, M. Jensen, Y. Lin, A.L. Lister, P. Lord, J. Malone, E. Manduchi, M. McGee, N. Morrison, J.A. Overton, H. Parkinson, B. Peters, P. Rocca-Serra, A. Ruttenberg, S.-A. Sansone, R.H. Scheuermann, D. Schober, B. Smith, L.N. Soldatova, C.J. Stoeckert Jr., C.F. Taylor, C. Torniai, J.A. Turner, R. Vita, P.L. Whetzel and J. Zheng, The Ontology for Biomedical Investigations, *PLOS ONE* **11**(4) (2016), 1–19. doi:10.1371/journal.pone.0154556. https://doi.org/10.1371/journal.pone.0154556.

[18] J.R. Quinlan, Induction of decision trees, *Machine Learning* **1**(1) (1986), 81–106, ISSN 1573-0565. doi:10.1007/BF00116251. https://doi.org/10.1007/BF00116251.

[19] J.R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1-55860-238-0.

[20] I.H.W. Eibe Frank Mark A. Hall, *The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques"*, Morgan Kaufmann, Fourth Edition, 2016.

[21] J. Platt, Fast Training of Support Vector Machines using Sequential Minimal Optimization, in: *Advances in Kernel Methods - Support Vector Learning*, B. Schoelkopf, C. Burges and A. Smola, eds, MIT Press, 1998. http://research.microsoft.com/\texttildelowjplatt/smo.html.

[22] L. Breiman, Random Forests, *Machine Learning* **45**(1) (2001), 5–32, ISSN 1573-0565. doi:10.1023/A:1010933404324. https://doi.org/10.1023/A:1010933404324.

[23] P. Panov, L.N. Soldatova and S. Džeroski, Towards an ontology of data mining investigations, in: *International Conference on Discovery Science*, Springer, 2009, pp. 257–271.

[24] R.R. Brinkman, M. Courtot, D. Derom, J.M. Fostel, Y. He, P. Lord, J. Malone, H. Parkinson, B. Peters, P. Rocca-Serra, A. Ruttenberg, S.-A. Sansone, L.N. Soldatova, C.J. Stoeckert, J.A. Turner, J. Zheng and the OBI consortium, Modeling biomedical experimental processes with OBI, *Journal of Biomedical Semantics* **1**(1) (2010), 7, ISSN 2041-1480. doi:10.1186/2041-1480-1-S1-S7. https://doi.org/10.1186/2041-1480-1-S1-S7.

[25] L.N. Soldatova and R.D. King, An ontology of scientific experiments, *Journal of the Royal Society Interface* **3**(11) (2006), 795–803.

[26] D. Garijo and Y. Gil, Augmenting PROV with Plans in P-PLAN: Scientific Processes as Linked Data, in: *Second International Workshop on Linked Science: Tackling Big Data (LISC), held in conjunction with the International Semantic Web Conference (ISWC)*, Boston, MA, 2012. http://www.isi.edu/~gil/papers/garijo-gil-lisc12.pdf.

[27] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan and J.V. den Bussche, The Open Provenance Model Core Specification (V1.1), *Future Gener. Comput. Syst.* **27**(6) (2011), 743–756, ISSN 0167-739X. doi:10.1016/j.future.2010.07.005. http://dx.doi.org/10.1016/j.future.2010.07.005.

[28] P. Missier and L. Moreau, PROV-DM: The PROV Data Model, W3C Recommendation, W3C, 2013, http://www.w3.org/TR/2013/REC-prov-dm-20130430/.

[29] S. Sahoo, T. Lebo and D. McGuinness, PROV-O: The PROV Ontology, W3C Recommendation, W3C, 2013, http://www.w3.org/TR/2013/REC-prov-o-20130430/.

[30] H. Hanke and D. Knees, A phase-field damage model based on evolving microstructure, *Asymptotic Analysis* **101** (2017), 149–180.

[31] K. Belhajjame, J. Zhao, D. Garijo, M. Gamble, K. Hettne, R. Palma, E. Mina, O. Corcho, J.M. Gómez-Pérez, S. Bechhofer et al., Using a suite of ontologies for preserving workflow-centric research objects, *Web Semantics: Science, Services and Agents on the World Wide Web* **32** (2015), 16–42.

[32] E. Lefever, A hybrid approach to domain-independent taxonomy learning, *Applied Ontology* **11**(3) (2016), 255–278.

[33] P.S. Meltzer, A. Kallioniemi and J.M. Trent, Chromosome alterations in human solid tumors, in: *The Genetic Basis of Human Cancer*, B. Vogelstein and K.W. Kinzler, eds, McGraw-Hill, New York, 2002, pp. 93–113.

[34] P.R. Murray, K.S. Rosenthal, G.S. Kobayashi and M.A. Pfaller, *Medical Microbiology*, 4th edn, Mosby, St. Louis, 2002.

[35] E. Wilson, Active vibration analysis of thin-walled beams, PhD thesis, University of Virginia, 1991.