# Towards Generating Automatic Machine Learning Metadata

Diego Esteves
AKSW, University of Leipzig
Germany
esteves@informatik.uni-leipzig.de

Pablo N. Mendes
IBM Research Almaden
California, USA
pnmendes@us.ibm.com

Diego Moussallem
AKSW, University of Leipzig
Germany
moussallem@informatik.uni-leipzig.de

Julio Cesar Duarte
Military Institute of Engineering (IME)
Rio de Janeiro, Brazil
duarte@ime.eb.br

Maria Claudia Cavalcanti
Military Institute of Engineering (IME)
Rio de Janeiro, Brazil
yoko@ime.eb.br

Jens Lehmann
University of Bonn
Germany
jens.lehmann@cs.uni-bonn.de

## ABSTRACT

Despite recent efforts to achieve a high level of interoperability of Machine Learning (ML) experiments, positively collaborating with the Reproducible Research context, we still run into the problems created due to the existence of different ML platforms: each of those have a specific conceptualization or schema for representing data and metadata. This scenario leads to an extra coding-effort to achieve both the desired interoperability and a better provenance level as well as a more automatized environment for obtaining the generated results. Hence, when using ML libraries, it is a common task to re-design specific data models (schemata) and develop wrappers to manage the produced outputs. In this article, we discuss this gap focusing on the solution for the question: “What is the cleanest and lowest-impact solution to achieve both higher interoperability and provenance metadata levels in the Integrated Development Environments (IDE) context and how to facilitate the inherent data querying task?”. We introduce a novel and low impact methodology specifically designed for code built in that context, combining semantic web concepts and reflection in order to minimize the gap for exporting ML metadata in a structured manner, allowing embedded code annotations that are, in run-time, converted in one of the state-of-the-art ML schemas for the Semantic Web: the MEX Vocabulary.

## Keywords

Machine Learning Outputs, Metadata, MEX, Reflection, Annotation, Interoperability, Provenance, Reproducible Research

## 1. INTRODUCTION

Machine Learning (ML) solutions have gained substantial attention as general workhorse solutions to a number of problems. For many problems there are several applicable algorithms, and it is not always clear from the start which algorithms will perform best. Much of the work when developing ML solutions goes to data preparation, feature extraction and parameter tuning. Different configurations will often produce very different results. It is possible, for example that two algorithms may "win" or "lose" to each other at first, but with different configurations they would trade places as winner/loser.

Managing the configurations, inputs and outputs of ML algorithms poses a huge challenge for developers. Many groups develop in house frameworks for managing their workflows, resulting in redundancy and increased maintenance costs (often within the same institution). Furthermore, when sharing experiment results, researchers often describe them with different language writing style (which is possible to become ambiguous) in their manuscripts making it difficult to directly compare results from different papers.

The MEX vocabulary [1] has the objective of describing ML algorithms and experiments to alleviate those problems. In this paper, we propose a methodology designed to inject MEX descriptions in the programming environment used by developers to run experiments. Our method is designed to minimize the gap for exporting ML metadata in a structured manner while imposing low development overhead. The method is built upon semantic web concepts, *annotations* and *reflection*.

As a result of using our methodology, results are better interpretable, research is more reproducible and managing experiments becomes easier. Our current implementation captures the ML algorithms' calls and can be directly used with the Java programming language. We have shared our implementation as an open source[1] project.

This paper is structured as follows: Section 2 progresses to discuss the *trade-off* problem regarding *interoperability, provenance and data management* existing among different levels of possible implementations (*SWFS*, *MLF* and *MLL*). Section 3 introduces an increasingly growing research area named *Reproducible Research* which benefits from our methodology. Section 4 presents related works in the area. Section 5 details the proposed architecture and presents the methodology, introducing the core and related concepts. Section 6 describes an implementation as a proof of concept, detailing Java use cases for the proposed methodology, as well as state its limitations. Section 7 discusses some issues regarding the challenges for achieving a maximum level of interoperability. Finally, Section 8 concludes the paper and introduces future works.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

[1] https://github.com/AKSW/mexproject

# 2. SWFS, MLF OR MLL: A TRADE-OFF PROBLEM

Machine learning has become an important tool for data scientists in research and business contexts. Plenty of workbenches/environments (*MLF*), libraries (*MLL*)[2] and workflow systems (*SWFS*) have emerged to serve as platform for creating ML models and executing experiments. Each of these provide a different level of implementation. Whereas *SWFS* provides a good level of provenance metadata, control execution and allows the interchange of experiment configurations between researchers that use the same tool, they lead to a high level of dependency and the configurations are not portable among other *SWFS* implementations. Moreover, they imply a high level of algorithm's implementation dependency (once only available algorithms can be used). Primarily, they are not commonly designed for specifically dealing with ML problems, but have either general scientific workflow proposes [2] or too specific scientific workflows [3] as focus of their implementation. Therefore, *SWFS* have the drawback which stands in the obligation of developing the solution specifically following its rules and natural limitations.

A further alternative, *MLFs* are specifically designed to deal with ML problems and commonly provide a broad range of ML algorithm implementations. Some of them allow experiment configurations [4] and do not require refined programming skills with its user interface. As drawbacks, we can mention the lack of provenance and interoperability among implementations. Also, this kind of platform does not allow programming flexibility to the user, reason why *APIs* (*MLL*) are often released in order to be loaded into IDEs for developing specific and flexible applications. Table 1 lists the characteristics and the main differences among each platform. Also, Figure 1 depicts examples for the 3 different platforms discussed.

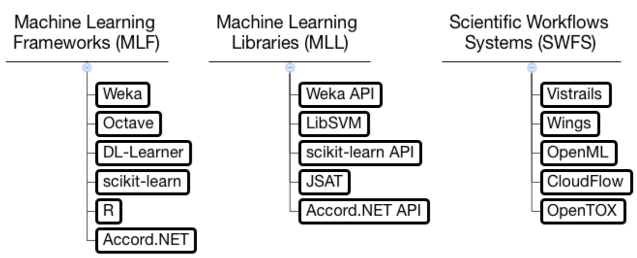


Figure 1: Examples of common machine learning platforms: *frameworks* that often implement a *front-end* interface (*MLF*), libraries to be imported into IDEs (*MLL*) and *workflow systems* which commonly have ML components as features (*SWFS*).

### 2.0.1 MLL: The Current Gap and Recurrent Solutions

As introduced, the major problem in the *MLL* context refers to the lack of *interoperability* and *provenance* metadata. Disregarding the possible lack schema problem, the *MLL* context also does not provide *data management* features, i.e., without a proper *management system* becomes tricky to get and analyze different dimensions of the generated data. Figure 2 exemplifies it, with an existing problem during the analysis step.

As a result, the lack of automatized and straightforward solutions for *data management* requires to develop *wrappers* and implement

[2]from this point on we are going to refer to *MLL* as the situation where a developer works with an API by importing it into an IDE, instead of just refer to the library itself.

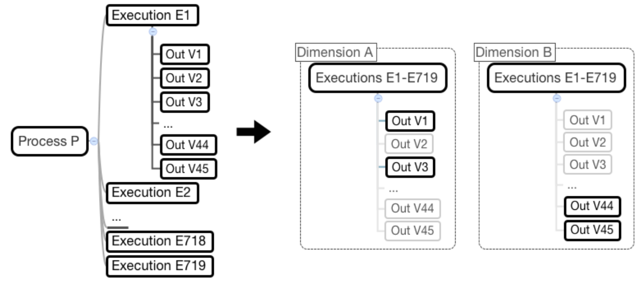


Figure 2: A common scenario representing a set of ML executions (*E1, E2, ..., E719*), started by a process *P*, which has its set of inputs and produces its outputs data (*V1 to V45*). Two dimensions are desired, one for obtain both the average of the *accuracy* (*V1*) and *error* (*V43*) (*Dimension A*) and other to count the set of entities that either fits to some specific rule A (*V44*) or B (*V45*) (*Dimension B*). In an example of large amount of data being produced outside a *workflow* system, a good question is how to query and extract specific parts of the generated output dataset, clustering different dimensions, in order to obtain the desired information as easy as possible without a *SGBD*.

*connectors* for some *DBMS*, for instance. This extra step brings the focus out of the main problem being investigated, i.e., an extra code-effort is required to set up the desired environment. On the other hand, avoiding this stage means to deal with pure *text files* or *stdout* outputs, which are not the best machine-readable solution and require a high level of effort to process and extract data, in addition to the discussed lack of *provenance* and *interoperability* (Figure 3). In other words, both situations are not welcome in terms of implementation effort.

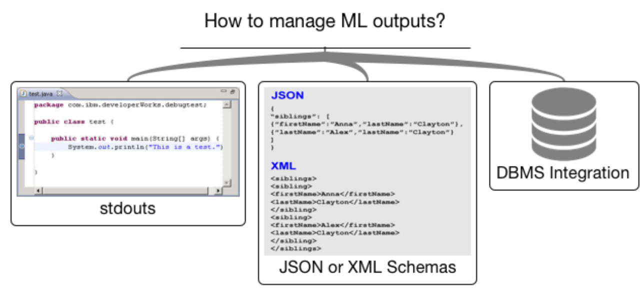


Figure 3: Managing output of machine learning executions in *MLL*: pure text (*stdouts*), self-schema definitions (e.g.:JSON or XML) or data base integrations (DBMS)

The *provenance* normally limits itself to some piece of documents written in natural language linked to the produced data. *Interoperability* issues are commonly treated with *self-schema* definitions, which are then shared among developers, e.g., by designing a particular simple structure using an existing standard (e.g.: JSON[3] or XML[4]) or just logging using an API (e.g.: LOG4J[5]). However, this scenario have 1) the inconvenience to present a poor level of metadata 2) an inability to represent semantically the data, abstracting specific implementation issues (e.g.: "logit function" and "logistic regression", which points out to the same concept) 3) the extra code-effort needed. Here, the SW comes into play, offering a much more sophisticated approach to achieve a higher level of provenance, but still allowing to achieve a decent level of interop-

[3]http://www.json.org

[4]http://www.w3schools.com/XML/xml_whatis.asp

[5]http://logging.apache.org/log4j/2.x/

| Platform | Advantages | Drawbacks |
|---|---|---|
| SWFS | High Provenance<br>Interoperability<br>Workflow Management | No (High) Interoperability<br>Dependency of the tool for updates |
| MLF | Front-end<br>No updates delay<br>(Low) Workflow Management | No (High) Interoperability<br>No much code flexibility |
| MLL | High code-flexibility | Low Provenance<br>Low Interoperability |

Table 1: Comparison of Machine Learning Platforms: Drawbacks and Advantages

erability. Endorsed by W3C, the RDF "*has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed*"[6]. In this paper we have developed a new methodology combining SW tools, *annotations* and *reflection* in order to reduce the effort to generate good and interoperable metadata as well as to provide query features. Table 2 summarizes the different strategies discussed to bridge the gap.

# 3. REPRODUCIBLE RESEARCH

A relatively recent *key term* to face this lack of metadata is *Reproducible Research*, which aims to make analytic data and code free available so that others will be able to reproduce findings, i.e., an environment where "provenance metadata" is accessible and a "high interoperability" level is achievable, so anyone is able to reproduce scientific achievements. Therefore, *Reproducibility* is one of the main principles of the scientific methods (Figure 4). According to the *IOM Report* [5] the following rules should be applied: 1) data/metadata publicly available; 2) the computer code and all the computational procedures should be available; 3) ideally the computer code will encompass all of the steps of computational analysis. The proposed work introduced in this paper aims to minimize the existing gap in this field by providing a methodology to automatic represent data collected from machine learning execution contexts, i.e., helping the representation of a subset of the required premises to achieve a complete reproducible environment.

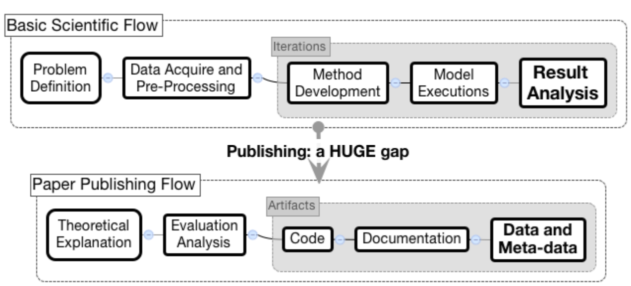


Figure 4: Experiments are hard to reproduce, when not impossible. Metadata is needed!

Figure 5 shows the evolution of the metadata generation for ML processes, from the poorest level possible (1) (e.g.: *plain text with no metadata at all*) until the maximum level of reproducible research possible (6). The second rounded rectangles ("*Schema Self-Definition*") emphasis on the most accomplished task when developing in the *MLL* context. Due the number of ML libraries available and lack of standards, developers tend to re-define schemas

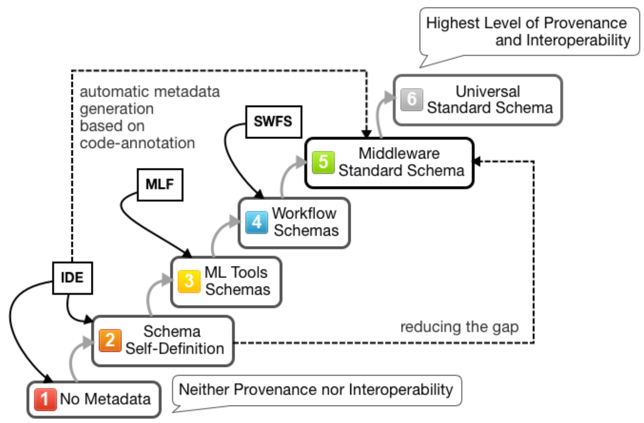


Figure 5: The evolution of metadata generation process in machine learning processes. The dashed arrow represents the contribution of this work

that, in the end, aim to share the same meaning. Third and fourth rounded rectangles ("*ML Tools*" and "*Workflow Schemas*", respectively) represent the further levels of ML implementations (*MLF* and *SWFS*) which provide different schema definitions existing in a less flexible environment. The item "*Middleware Standard Schema*" (5) highlights the proposed model based on vocabularies in order to provide a high-level model to exchange machine learning output metadata. Finally, the last rounded rectangle ("*Universal Standard Schema*") represents the best scenario possible, where every ML platform (tool/framework/library) exports common variables in a standard manner. We argue that this scenario is not achievable due to political conflicts, complexity of scenarios and extra amount of work needed.

# 4. RELATED WORK

To the best of our knowledge this is the first report about a methodology to support the automatic generation of metadata for machine learning executions in *MLL* contexts based on a common vocabulary. As introduced before (Section 1 and Section 2), different platforms coexist to use the ML algorithms (*MLL*, *SWFS* and *MLF*) but all of them fail in abstracting the concepts behind ML, mainly focused in the run of each algorithm in a simple manner, generating a *free format* that can be interpreted regardless strategy of implementation. As a consequence, positively collaborating along with *Reproducible Research*. A slightly similar approach is the *knitr*[7], a markup-language that allows embedded annotations, for generating dynamic reports though.

[6] http://www.w3.org/RDF/

[7] http://yihui.name/knitr/

| Method | Advantages | Drawbacks |
|---|---|---|
| *stdout* | No Extra Coding Effort Required | Lack of Provenance<br>Lack of Interoperability<br>Lack of Data Query Feature |
| DBMS | Data Query Feature | Extra Coding Effort (Integration)<br>Lack of Provenance<br>Lack of Interoperability |
| Self-schema Definition | Straightforward Solution | Extra Coding Effort<br>Extra Analysis Effort (modeling)<br>Lack of Provenance<br>Lack of Interoperability |
| Annotations + SW | Provenance<br>Interoperability<br>Data Query Feature<br>Automatic Metadata Generation | Extra Processing Time<br>Security Issues |

Table 2: Comparison of strategies for representing machine learning metadata in *MLL* contexts

# 5. THE ARCHITECTURE: A NOVEL LOW-IMPACT APPROACH FOR METADATA GENERATION AND DATA OUTPUT MANAGEMENT IN MLL CONTEXTS

In this section we explain the proposed architecture and briefly introduce related data models that could further extend the methodology. The major contribution is to allow metadata generation regardless *IDE*, machine-learning *library* and context of the problem. Figure 6 depicts the general process of generating the metadata. In this example, two annotated Java classes following the MEX annotation's descriptions are passed by parameter to the *MetaGeneration* class. The entire process occurs in a transparent manner and no further step is required (Listing 4 exemplifies the process). By doing so, developers reach a clean solution to narrow down the issues discussed before (Section 2.0.1)

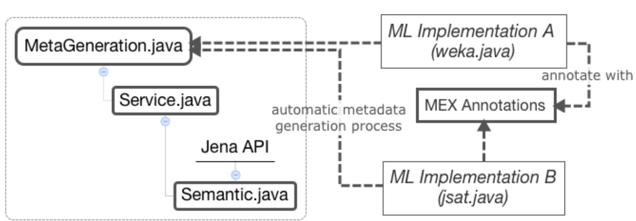


Figure 6: MEX Framework at a glance: a low impact solution for generating machine learning metadata from Java classes

The produced metadata is based on the MEX [1], a vocabulary designed specifically to deal with outputs of machine learning executions and relies in three main layers: *mexcore*[8] for execution's controlling, *mexalgo*[9] for ML algorithms representations and *mexperf*[10] for performance indicators. It is a lightweight format built upon on W3C PROV-O[11] - categorized as a vocabulary - which abstracts the core machine learning concepts regarding the execution of an algorithm (Figure 7). Whereas OntoDM [6], Exposé [7] and DMOP [8] are classified as Ontologies and were designed for representing data mining workflows, the Predictive Model Markup Language (PMML) [9] is a XML based schema and was conceived to represent (predictive and descriptive) data models as well as pre and post-processing. In this scenario, MEX stands as a flexible and lightweight solution for representing the basic triple - *inputs*, *run* and *outputs* - for any machine learning algorithm. Figure 8 depicts current technologies and schemas for representing machine learning metadata.

[8]http://mex.aksw.org/mex-core
[9]http://mex.aksw.org/mex-algo
[10]http://mex.aksw.org/mex-perf
[11]http://www.w3.org/TR/prov-o/

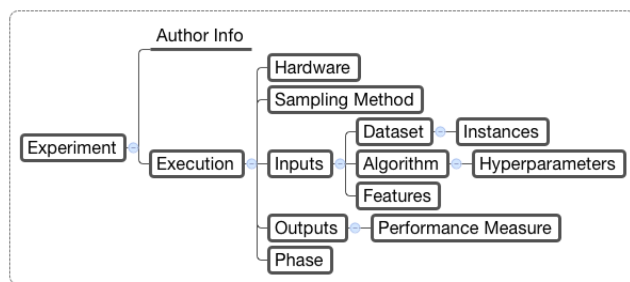


Figure 7: MEX structure: the core for representing machine learning executions as simple as possible

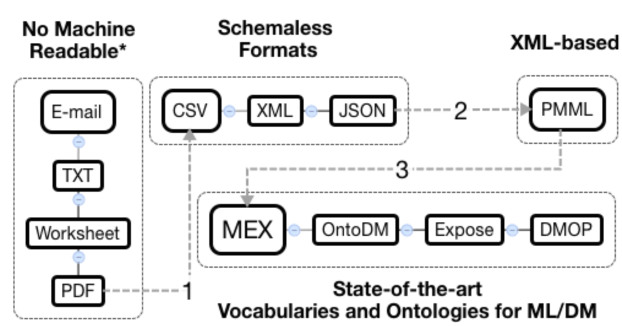


Figure 8: Open-source formats for representing ML metadata: from straightforward representations (1) formats until more refined schema representations (3).

## 5.1 Reflection and Annotations

Reflection is a widely used technique that allows software to manipulate applications by inspecting variables or altering its run-time behavior. For instance, in Java, reflection can be used through its virtual machine, allowing the inspection of interfaces, classes, methods and data attributes at run-time. Listing 1 show an example of a reflected code.

```
static Object callMethod(Object o, Object v) throws Exception{
    Method[] methods = o.getClass().getDeclaredMethods();
    for (final Method method : methods) {
        MethodAnn a = method.getAnnotation(MethodAnn.class);
        if(a!=null) method.invoke(object, value);}}
```

Listing 1: Invoking a method with reflection

Annotations, on the other hand, have no effect on the execution of the program but provides metadata about itself. Annotations are usually preceded by a @-symbol and indicate an auxiliary information that can be captured both at compile and execution time. In Java, they are quite similar to Javadoc tags, although they can be reflective and accessed by the Virtual Machine. Listing 2 exemplifies its usage.

```
@MachineLearningExperiment
public class NaiveBayes extends MLAlgorithm{
@model Model m;
@trainProcedure public void train(Dataset d){...}
@classProcedure public void classify(Dataset d){...}
```

Listing 2: Annotations: metadata for Java classes

The annotation interface follows the MEX Vocabulary structure. Table 3 details the existing MEX annotations. The complete documentation can be founded at the project's website.

## 5.2 Query Templates with SPARQL: Making the Information Extraction Process Easier

An often task when dealing with a machine learning problem is the large amount of produced data. A simple run can produce many variables that will be further analyzed (Figure 2). Since *IDEs* does not provide a standard manner to store these data, a recurrent solution is to design a new schema for representing the metadata and implement a *wrapper* for connecting and storing the information, which have the inconvenient to be technology-dependent, besides the extra-effort needed for coding *connectors*, as discussed in the Section 2. The *RDF* has the advantage to allow querying with *SPARQL* commands (Listing 3). A developer could, for instance, search for the best executions of an *SVM* model by just uploading the *mex files* into a triple-store[12] database. No extra development is required. The meta file is ready to be consumed.

```
PREFIX mexcore: <http://mex.aksw.org/mex-core/>
PREFIX mexperf: <http://mex.aksw.org/mex-perf/>
PREFIX mexalgo: <http://mex.aksw.org/mex-algo/>
SELECT ?e ?acc
WHERE { ?e mexperf:accuracy ?acc .
        FILTER (?acc > 0.90)
        ?e prov:used mexalgo:C-SVM . }
```

Listing 3: A generic SPARQL query: looking for best configurations for C-SVM executions

[12]https://jena.apache.org/documentation/tdb/

## 5.3 Drawbacks and Limitations

Despite a more clean and less coupled solution (once a vocabulary provides a context-less list of common terms), the proposed methodology face some limitations, as follows:

- *Reflection and Annotations*: programming-language must allow reflection and annotations. As use case, we have implemented *Java* examples, although other programming languages could be used (as long as it implements reflection).
- *Performance Overhead and Security Restrictions*: the use of *Reflection* directly impacts on the execution-time, decreasing the overall performance as well as expose the code impacting in security restrictions[13].
- *Methodology Coverage*: The MEX Vocabulary covers just pure machine learning metadata (an algorithm, its inputs and outputs for given execution). Pre-processing steps or data mining tasks are not covered due the complexity of the task.
- *Local Variables*: *reflection* in *Java* does not allow to capture local variables, i.e., variables that are not explicitly declared as *class variables* can not be obtained via *annotations* and *reflection*.

## 5.4 Advantages

The biggest benefit of the proposed methodology is to use a standard model which abstracts the particular concepts existing into each ML environment/implementation and to create an upper layer that can be able to inter connect knowledge as easy as possible with the produced metadata. The following list details the key advantages:

- *In-line Annotations*: a *Java* class can be simply annotated and the metadata will be generated in *run-time*.
- *More Abstraction*: by using a vocabulary, developers can benefit of the high level of abstraction provided. A *Support Vector Machines* algorithm for a classification problem can be represented with a single reference: `http://mex.aksw.org/mex-algo#C-SVM`, there is no need to redefine a vocabulary.
- *Less Coding Effort and More Agreement Rate*: there is no need to create and share the structure of the schema for representing the output data.
- *Better Interoperability and Provenance Levels*: a common schema allows higher levels of data interchanging and *RDF* encourages better metadata descriptions.
- *Querying Capabilities*: Once the vocabulary is RDF-based, developers can benefit of SPARQL queries[14].
- *Reproducible Research*: the methodology collaborates with reproducible research rules, following best practices for data publishing and code management.

[13]https://docs.oracle.com/javase/tutorial/reflect/
[14]http://www.w3.org/TR/rdf-sparql-query/

| MEX Layer | MEX Annotation | Description |
|---|---|---|
| :mexcore | @Execution | the run (training or test phase) |
| | @Dataset | the dataset name |
| | @Feature | the features |
| | @StartTime | the process start time |
| | @EndTime | the process end time |
| | @TrainingProcedure | the training method |
| | @TestProcedure | the test method |
| | @Experiment | the authoring info |
| :mexalgo | @Algorithm | the ML algorithm |
| | @Hyperparameter | the ML algorithm's parameter |
| :mexperf | @Measure | the performance measure |
| N/A | @Start | the main method to be executed |

Table 3: Examples of MEX Annotations for Java code: the highest level of abstraction to export metadata of an ML algorithm's execution.

## 6. PROOF OF CONCEPT

In order to automatic export the metadata, the provided Java class has to be annotated following the MEX annotations interface (Table 3). The metadata generation process then occurs transparently with *reflection*, which executes the user class (*IrisWekaExample.java*) and maps to the vocabulary in *run-time* (Listing 4), generating the metadata file (*mymex01.ttl*). As mentioned, the proposed approach does not require self-schema definitions or extra coding-effort to create *wrappers* for *SGBDs*, for instance. It also provides query feature with SPARQL. With this novel concept, just *Java* annotations following the MEX structure are required. In this example, we run a *J48* algorithm implementation over the iris dataset[15].

```
java -cp /home/user/mexframework org.aksw.mex.framework.MetaGeneration -uc IrisWekaExample.java -out mymex01.ttl
java -cp /home/user/mexframework org.aksw.mex.framework.MetaGeneration -uc IrisJSATExample.java -out mymex02.ttl
```

Listing 4: MEX Framework usage: starting the automatic metadata generation

```
this:ml1 a prov:Entity , mexperf:ClassificationMeasure , mexperf:RegressionMeasure;
  dct:identifier "WekaPerformance";
  mexperf:accuracy "0.9768"^^xsd:float;
  mexperf:truePositive "147"^^xsd:integer;
  mexperf:falsePositive "3"^^xsd:integer;
  mexperf:kappaStatistics "0.97"^^xsd:float;
  mexperf:meanAbsoluteError "0.0233"^^xsd:float;
  mexperf:rootMeanSquaredError "0.108"^^xsd:float;
  mexperf:relativeAbsoluteError "0.052482"^^xsd:float;
  mexperf:rootRelativeSquaredError "0.0229089"^^xsd:float;
  prov:wasGeneratedBy this:ep1;.
```

Listing 5: An excerpt of the generated metadata file

Unlike *stdouts*, we achieve now a much better machine-readable structure, able to perform queries and inter-connect experiments. The following text (Listing 6) depicts the default *Weka* (performance measures) outputs for the given execution whereas the Listing 5 represents the produced metadata.

```
=== Evaluation on training set ===
=== Summary ===
Correctly Classified Instances        147      98%
Incorrectly Classified Instances        3       2%
Kappa statistic                         0.97
Mean absolute error                     0.0233
Root mean squared error                 0.108
Relative absolute error                 5.2482%
Root relative squared error            22.9089%
Total Number of Instances             150
```

Listing 6: An excerpt of the default Weka *stdout*: how to query and interchange the generated metadata?

Besides, an important advantage is the high level of interoperability. In order to highlight it we have used the *JSAT API (Java Statistical Analysis Tool)*[16] to execute the same task, but with a *Naive Bayes* model. An excerpt of the output is summarized as follows (Listing 7)

```
There are 5 features for this data set.
1 categorical features
They are:
  class
4 numerical features
They are:
  sepallength
  sepalwidth
  petallength
  petalwidth
...
146| True Class: 2, Predicted: 2, Confidence: 0.9745542454188852
147| True Class: 2, Predicted: 2, Confidence: 0.9996298333223543
148| True Class: 2, Predicted: 2, Confidence: 0.9999997539798201
149| True Class: 2, Predicted: 2, Confidence: 0.9439950025737772
6 errors were made, 4.0% error rate
```

Listing 7: An excerpt of the default JSAT *stdout*: very different logging structure for the same task

As result, we end up with a much more (machine-readable) interoperable structure. Now, two different ML algorithms (*Naive*

[15] https://archive.ics.uci.edu/ml/datasets/Iris

[16] https://github.com/EdwardRaff/JSAT

*Bayes* and *J48*) used with different Java libraries can be compared, stored and interpreted (Listings 5 and 8).

```
this:ml1 a prov:Entity , mexperf:
    ClassificationMeasure;
  dct:identifier "JSATPerformance";
  mexperf:accuracy "0.96"^^xsd:float;
  ...
  prov:wasGeneratedBy this:ep1;.
```

Listing 8: An excerpt of the generated metadata file

More technical details can be founded at the project website[17].

## 7. DISCUSSION

- *Universal Schema for ML/DM*: ML is a growing research topic and over and over developers have been obtaining access to different tools, methods and algorithms to solve specific problems. This evolution leads to a complex scenario for data analysis and manipulation. In that sense, people should agree in a common format for interchanging and manipulating data. A common format, however, is most likely to be unachievable. Companies, open-source tools and developers would have to agree in a common data process generation for each specific task with can be considered as an utopian scenario. A recent effort in the semantic web community stands for achieving a reasonable level of mapping among state-of-the-art vocabularies and ontologies for machine learning and data mining tasks[18][19].

- *Logging and Data Management*: we aim to provide a new methodology that facilitates the management of simple ML outputs in *MLL* contexts, i.e., one or more ML libraries being manipulated into an *IDE*. As discussed, this kind of environment lacks of a standard in order to export metadata and developers and scientists tend to do not care about metadata generation, once it cause an extra effort to define, develop and export the data by implementing some pre-defined structure.

- *100% automatic process*: a more interesting approach, e.g.: a totally transparent metadata generation process, however, also tends to be cumbersome to implement. Coding embedded methods in the machine-learning libraries (e.g.: *Weka API*) or machine-learning frameworks (e.g.: *Octave*) are both extremely laborious as well as error-prone, once it becomes dependent of the given library/workbench, i.e., an update in its interfaces requires updates in the implemented code.

- *Natural Resistance*: reproducible research issues are hard to follow due natural resistance of producing data with high level of data quality. Researchers and developers tend to avoid extra-work due schedule and budget limitations and get satisfied with simple unstructured reports. The importance of such metadata is clear for most of the people just in long-term scenarios, such as *meta-learning* analysis and the achievement of a more automatize programming environment, for instance.

Therefore, our method aims to approximate to the more plausible solution, which combines the flexibility of *MLL* scenarios and some of the features of the *SWFS*, such as good interoperability and provenance levels, whereas aims to narrow the gap between current approach to deal with ML outputs and good standards of reproducible research. Furthermore, a model built up on *RDF* allows data querying, which also becomes a very interesting feature for researchers.

[17]https://github.com/AKSW/mexproject
[18]https://www.w3.org/community/ml-schema/
[19]https://github.com/ML-Schema/core/wiki

## 8. CONCLUSIONS AND FUTURE WORK

In this paper we have discussed the current gap on generating ML metadata in different platforms and introduced a new methodology for exporting metadata of ML executions in *MLL* scenarios, bridging the gap between reproducibility issues and ML scripts. As a proof of concept, we have demonstrated examples of this new methodology built upon SW concepts. We argue that this approach minimizes the code-effort avoiding extra developments and makes the code more clean. Also, a very useful advantage is the easy manipulation of the produced output data, which can be queried with *SPARQL* language, adding flexibility to the data manipulation task. As future work we plan 1) to integrate more sophisticated ontologies in that methodology (e.g.:DMOP [8]) in order to also cover DM scenarios and 2) to design a more robust framework that allows to schedule runs of algorithms with different configurations, i.e., automatic pipelines. Here, the annotations would be mapped to be instantiated in *run-time* with dynamic parameters defined beforehand by the user, e.g.: calling different algorithms with different hyper-parameter (*SVM*, *Naive Bayes* and *Regression Logistic*) in parallel to perform the same task, sharply automating the development environment for *MLL* context.

## 9. ADDITIONAL AUTHORS

Additional author: Ciro Baron Neto (University of Leipzig, Germany, email: `cbaron@informatik.uni-leipzig.de`) and Igor Costa (Military Institute of Engineering (IME), email: `igorsc@ime.eb.br`).

## 10. REFERENCES


[1] Diego Esteves, Diego Moussallem, Ciro Baron Neto, Tommaso Soru, Ricardo Usbeck, Markus Ackermann, and Jens Lehmann. MEX Vocabulary: A Lightweight Interchange Format for Machine Learning Experiments. In *Proceedings of the 11th International Conference on Semantic Systems*, pages 169–176. ACM, 2015.

[2] Steven P. Callahan, Juliana Freire, Emanuele Santos, Carlos E. Scheidegger, Cláudio T. Silva, and Huy T. Vo. Vistrails: Visualization meets data management. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 745–747, New York, NY, USA, 2006. ACM.

[3] Olga Tcheremenskaia, Romualdo Benigni, Ivelina Nikolova, Nina Jeliazkova, Sylvia Escher, Monika Batke, Thomas Baier, Vladimir Poroikov, Alexey Lagunin, Micha Rautenberg, et al. Opentox predictive toxicology framework: toxicological ontology and semantic media wiki-based opentoxipedia. *J. Biomedical Semantics*, 3(S-1):S7, 2012.

[4] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[5] Christine M Micheel, Sharly J Nass, Gilbert S Omenn, et al. *Evolution of translational omics: lessons learned and the path forward*. National Academies Press, 2012.

[6] Panče Panov, Larisa Soldatova, and Sašo Džeroski. Ontodm-kdd: Ontology for representing the knowledge discovery process. In *Discovery Science*, pages 126–140. 2013.

[7] Joaquin Vanschoren and Larisa Soldatova. Exposé: An ontology for data mining experiments. *International workshop on third generation data mining: Towards service-oriented knowledge discovery (SoKD-2010)*, pages 31–46, 2010.

[8] C. Maria Keet, Agnieszka Ławrynowicz, Claudia d'Amato, Alexandros Kalousis, Phong Nguyen, Raul Palma, Robert Stevens, and Melanie Hilario. The Data Mining OPtimization Ontology. *Web Semantics: Science, Services and Agents on the World Wide Web*, pages 43 – 53, 2015.

[9] Alex Guazzelli, Michael Zeller, Wen-Ching Lin, and Graham Williams. PMML: An open standard for sharing models. *The R Journal*, 1(1):60–65, June 2009.