MEX Interfaces: Automating Machine Learning Metadata Generation

Conference Paper · September 2016
DOI: 10.1145/2993318.2993320

8 authors, including:

Diego Esteves
University of Leipzig
8 PUBLICATIONS 4 CITATIONS

Amrapali Zaveri
Stanford University
29 PUBLICATIONS 240 CITATIONS

Jens Lehmann
University of Bonn
148 PUBLICATIONS 4,717 CITATIONS

M. C. Cavalcanti
Instituto Militar de Engenharia (IME)
50 PUBLICATIONS 181 CITATIONS

Some of the authors of this publication are also working on these related projects:

BigDataEurope View project
WDAqua ITN View project

All content following this page was uploaded by Diego Esteves on 09 December 2016.

The user has requested enhancement of the downloaded file. All in-text references underlined in blue are added to the original document and are linked to publications on ResearchGate, letting you access and read them immediately.
MEX Interfaces: Automating Machine Learning Metadata Generation

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

Amrapali Zaveri  
Stanford Center for Biomedical Informatics Research  
Stanford University, USA  
amrapali@stanford.edu

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 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 automated 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, i.e., the minimal effort 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: 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 into 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, the 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 project [2].

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 software implementations: Scientific Workflow Systems (SWFS), Machine Learning Frameworks (MLF) and Machine Learning Libraries (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. Sec-

1 https://docs.oracle.com/javase/tutorial/java/annotations/basics.html
2 https://docs.oracle.com/javase/tutorial/java/reflect/
3 https://github.com/AKSW/mexproject
forms discussed. Also, Figure 1 depicts examples for the three different platforms into IDEs for developing specific and flexible applications. Table 1 is a reason why the platform does not allow programming flexibility to the user, which is not commonly designed for specifically dealing with ML problems, but have either general scientific workflow proposed [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.

Another alternative, MLF are specifically designed to deal with ML problems and commonly provide a broad range of ML algorithm implementations. Some of them allow experiment configurations between researchers that use the same tool. However, 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 only once available algorithms can be used. Primarily, they are not commonly designed for specifically dealing with ML problems, but have either general scientific workflow proposed [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.

As introduced, the major problem in the MLL context refers to the lack of interoperability and provenance metadata. Disregarding the possible lack of 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.

As a result, the lack of automatized and straightforward solutions for data management requires to develop wrappers and implement connectors for any Database Management Systems (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 2). In other words, both situations are not welcome in terms of the implementation effort.

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

<table>
<thead>
<tr>
<th>Platform</th>
<th>Advantages</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWFS</td>
<td>High Provenance</td>
<td>No (High) Interoperability</td>
</tr>
<tr>
<td></td>
<td>Interoperability</td>
<td>Updates are dependent of tool</td>
</tr>
<tr>
<td></td>
<td>Workflow Management</td>
<td></td>
</tr>
<tr>
<td>MLF</td>
<td>Front-end</td>
<td>No (High) Interoperability</td>
</tr>
<tr>
<td></td>
<td>No updates delay</td>
<td>No much code flexibility</td>
</tr>
<tr>
<td></td>
<td>(Low) Workflow Management</td>
<td></td>
</tr>
<tr>
<td>MLL</td>
<td>High code-flexibility</td>
<td>Low Provenance</td>
</tr>
<tr>
<td></td>
<td>Low Interoperability</td>
<td></td>
</tr>
</tbody>
</table>

2.1 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 of 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.

As a result, the lack of automatized and straightforward solutions for data management requires to develop wrappers and implement connectors for any Database Management Systems (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 2). In other words, both situations are not welcome in terms of the implementation effort.

The provenance normally limits itself to an excerpt of text 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 [4] or XML [5]) or just logging using an API (e.g.: LOG4j [6]). However, this scenario has 1) the inconvenience to present a poor level of metadata 2) an inability to represent the data semantically, 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 interoperability. Endorsed by W3C, 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.” 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 inter-operable 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 freely 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 3). 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 automatically 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 3: Experiments are hard to reproduce, when not impossible. Standards and Metadata are needed!

Figure 4 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 steps (“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 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 of the strategy of implementation. As a consequence, positively collaborating along with Reproducible Research context. A slightly similar approach is the knitr, a markup-language that allows embedded annotations, for generating dynamic reports though.

5. MEX INTERFACES: A NOVEL LOW IMPACT APPROACH FOR METADATA GENERATION

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 of the IDE, machine-learning library and context of the experiment. We argue developers dealing with machine learning problems can directly benefit of the interfaces, automatizing the process of generating metadata of machine learning experiments. Furthermore, the proposed interfaces provide guidance on the standardization of the generated metadata, once they are based on a state of the art vocabulary for ML [10]. Figure 5 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 5 exemplifies the process). By doing so, developers reach a clean solution to narrow down the issues discussed before (Section 2).

The produced metadata is based on MEX [11], a vocabulary specifically designed to deal with inputs and outputs of machine learn-

http://www.w3.org/RDF/
http://yihui.name/knitr/
https://github.com/ML-Schema/core/wiki/Vocabulary
<table>
<thead>
<tr>
<th>Method</th>
<th>Advantages</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>stdout</td>
<td>No Extra Coding Effort Required</td>
<td>Lack of Provenance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lack of Interoperability</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lack of Data Query Feature</td>
</tr>
<tr>
<td>DBMS</td>
<td>Data Query Feature</td>
<td>Extra Coding Effort (Integration)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lack of Provenance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lack of Interoperability</td>
</tr>
<tr>
<td>Self-schema Definition</td>
<td>Straightforward Solution</td>
<td>Extra Coding Effort</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Extra Analysis Effort (modeling)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lack of Provenance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lack of Interoperability</td>
</tr>
<tr>
<td>Annotations + SW</td>
<td>Provenance</td>
<td>Extra Processing Time</td>
</tr>
<tr>
<td></td>
<td>Interoperability</td>
<td>Security Issues</td>
</tr>
<tr>
<td></td>
<td>Data Query Feature</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Automatic Metadata Generation</td>
<td></td>
</tr>
</tbody>
</table>

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

Figure 5: MEX Interfaces at a glance: a low impact solution for generating machine learning metadata from annotated classes

Figure 6: Open-source formats for representing ML metadata: from straightforward representations (1) formats until more refined schema representations (2)(3). Note: (*) Although it can be - technically - considered machine-readable, we assume that the effort to make it happen does not pays off.

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.

```java
static Object callMethod(Object o, Object v) throws Exception {
  // Method implementation
}
```

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 accessible by the Virtual Machine. Listing 2 exemplifies its usage.

```java
@MachineLearningExperiment
public class NaiveBayes extends MLAlgorithm {
  public void train(Dataset d) {
    // Training process
  }
  public void classify(Dataset d) {
    // Classification process
  }
}
```

The annotation interface follows the MEX Vocabulary structure. Table 3 details the existing MEX annotations. The complete docu-
Table 3: Examples of MEX Annotations for Java code: the highest level of abstraction to export metadata of an ML algorithm’s execution.

<table>
<thead>
<tr>
<th>MEX Layer</th>
<th>MEX Annotation</th>
<th>Description</th>
<th>Group of Annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>:mexcore</td>
<td>@Execution</td>
<td>the run (training or test phase)</td>
<td>Functions</td>
</tr>
<tr>
<td>:Dataset</td>
<td>@Algorithm</td>
<td>the ML algorithm</td>
<td>Functions</td>
</tr>
<tr>
<td>:Features</td>
<td>@Hyperparameter</td>
<td>the ML algorithm’s parameter</td>
<td>Functions</td>
</tr>
<tr>
<td>:TrainingProcedure</td>
<td>@Measure</td>
<td>the performance measure</td>
<td>Functions</td>
</tr>
<tr>
<td>:TextProcedure</td>
<td>@Experiment</td>
<td>the authoring info</td>
<td>Basic Provenance</td>
</tr>
<tr>
<td>:mexperf</td>
<td>@Measure</td>
<td>the performance measure</td>
<td>Functions</td>
</tr>
</tbody>
</table>

Table 4: Output of (Listing 3)

<table>
<thead>
<tr>
<th>executionID</th>
<th>algorithm</th>
<th>f1</th>
</tr>
</thead>
<tbody>
<tr>
<td>“C0_MEX_EXEC_D44”</td>
<td>“Bagging48”</td>
<td>0.9968</td>
</tr>
<tr>
<td>“C0_MEX_EXEC_D24”</td>
<td>“Logistic Model Trees”</td>
<td>0.9968</td>
</tr>
<tr>
<td>“C0_MEX_EXEC_D16”</td>
<td>“Random Forest”</td>
<td>0.9968</td>
</tr>
<tr>
<td>“C0_MEX_EXEC_D64”</td>
<td>“Multilayer Perceptron”</td>
<td>0.9967</td>
</tr>
</tbody>
</table>

5.2 Query Templates with SPARQL: Making the Data Management Process Easier

An often scenario when dealing with a machine learning problem is the large amount of produced data by the experiments. A simple run can produce many variables that will be further analyzed. Since IDEs do 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 produced information, which has the disadvantage to be technology-dependent, besides the extra-effort needed for coding connectors (as discussed in the Section 2). RDF has the advantage to allow querying with SPARQL commands (Listing 3). Therefore, a developer could, for instance, search for the best executions of a given model (e.g., svm) by just uploading the produced mex files into a triple-store. No extra development is required. The metadata file is ready to be consumed and readily present the desired information (Table 4).

```
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 DISTINCT ?executionID ?algorithm ?f1 
WHERE {
  ?exec prov: id ?executionID.
  ?exec prov: used ?alg.
  ?p prov: wasGeneratedBy ?exec.
  ?alg a mexalgo: Algorithm.
} 
ORDER BY DESC (?f1)
LIMIT 4
```

Listing 3: A generic SPARQL query: looking for the best top 4 configurations based on f1 scores.

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 faces some limitations, as follows:

- **Reflection and Annotations**: programming-language must allow reflection and annotations. As a use case, we have implemented Java examples, although other programming languages could be used (as long as it implements reflection). In case reflection is not allowed, LOG4MEX can be used for logging.

- **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. An impact analysis of performance is planned, although we argue that the most costly steps in ML scripts are I/O operations and mathematical calculations and not object creations.

- **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 to 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 cannot 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 is 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 from SPARQL queries.
- Reproducible Research: the methodology collaborates with reproducible research rules, following best practices for data publishing and code management.

6. PROOF OF CONCEPT

In order to automatically export the metadata, the provided (java) class has to be annotated following the MEX annotations interface (Table 3). Listing 4 depicts an excerpt of annotated class.

```java
public class WekaExample001 {

    @Hardware (cpu = MEXEnum.EnumProcessors.INTEL_CORE7, memory = MEXEnum.EnumRAM.SIZE_8GB, hdType = "SSD")
    @SamplingMethod (class = MEXEnum.EnumSamplingMethods.CROSS_VALIDATION, trainSize = 0.5, testSize = 0.5, folds = 10)
    @InterfaceVersion (version = MEXEnum EnumAnnotationInterfaceStyles.MI)
    public class WekaExample001 {

        private final static Logger LOG = Logger.getLogger (WekaExample001.class);

        @DatasetName public String ds = "iris.arff"; Instances data;

        Listing 4: An excerpt of annotated java class

        The metadata generation process then occurs transparently with reflection, which executes the class (IrisWekaExample.java) and maps to the vocabulary in run-time (Listings 5 and 6), generating the metadata file (mymex01.ttl).
```

Listing 6: An excerpt of log for the metadata creation process based on the interfaces

As mentioned, the proposed approach does not require self-schema definitions or extra coding-effort to create wrappers for DBMS, for instance. It also provides query feature with SPARQL (Section 5.4). With this novel concept, just (java) annotations following the MEX interfaces are required. In this example, we run a J48 algorithm implementation over the iris dataset.

Unlike stduots, we now achieve a much better machine-readable structure, which is able to perform queries and inter-connect experiments. The following output (Listing 7) represents the produced metadata whereas Listing 8 depicts the default Weka (performance measures) outputs for the given execution.

```
Listing 7: An excerpt of the generated metadata file
```

http://www.w3.org/TR/rdf-sparql-query/

https://archive.ics.uci.edu/ml/datasets/Iris
7. DISCUSSION

- Universal Schema for ML/DM: ML is a growing research topic where developers have been obtaining access to different tools, methods and complex algorithms to solve specific problems. This evolution leads to a complex scenario for data analysis and data manipulation. In that sense, people should agree on 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[11].

- 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 a standard to export metadata and developers and scientists tend to not care about metadata generation, as it cause an extra effort to define, develop and export the data by implementing some predefined 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 to natural resistance of producing data with high level of data quality. Researchers and developers tend to avoid extra-work due to schedule and budget limitations and get satisfied with simple unstructured reports. The importance of such metadata is clear for most of the people 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 provenance level and data management feature. Due to the adoption of a vocabulary, high interoperability is achieved. Thus, we aim 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.

8. CONCLUSIONS AND FUTURE WORK

In this paper we have discussed the current gap on generating ML metadata for 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 making 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 and provide features to convert metadata from one to another (e.g.: DMOP[8] to MEX[1] and vice-versa) in order to also cover DM scenarios. 2) To analyse the coverage of the methodology with more machine learning scenarios, such as
non-supervised algorithms and 3) to design a more robust framework that allows to schedule runs of algorithms with different configurations, i.e., automatic pipelines based on configuration files. Here, the annotations would be mapped to be instantiated in runtime 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.

Acknowledgments This work was supported by a grant from the EU H2020 Framework Programme provided for the project Big Data Europe (GA no. 644564) and HOBBIT (GA no. 688227). I also would like to thank CAPES foundation for the PhD scholarship (Ministry of Education of Brazil, Brasilia - DF 70040-020, Brazil - Bolsista da CAPES - Proc. n: BEX 10179/13-5)

9. ADDITIONAL AUTHORS

Additional authors: Ciro Baron Neto (University of Leipzig, Germany, e-mail: cbaron@informatik.uni-leipzig.de) and Igor Costa (Military Institute of Engineering - IME, Brazil, e-mail: igorsc@ime.eb.br) and Maria Claudia Cavalcanti (Military Institute of Engineering - IME, Brazil, e-mail: yoko@ime.eb.br)

10. REFERENCES


