A Unified Model Representation of Machine Learning Knowledge

J.G. Enríquez[1,*], A. Martínez-Rojas[1], D. Lizcano[2] and A. Jiménez-Ramírez[1]

[1] Computer Languages and Systems Department. Escuela Técnica Superior de Ingeniería Informática, Avenida Reina Mercedes, s/n, 41012, Sevilla. Spain.
[2] Universidad a distancia de Madrid. Carretera de La Coruña, KM.38,500, vía de Servicio, nº 15, 28400, Collado Villalba, Madrid. Spain.
[*] Corresponding author
E-mail: jgenriquez@us.es

Abstract

Nowadays, Machine Learning (ML) algorithms are being widely applied in virtually all possible scenarios. However, developing a ML project entails the effort of many ML experts who have to select and configure the appropriate algorithm to process the data to learn from, between other things. Since there exist thousands of algorithms, it becomes a time-consuming and challenging task. To this end, recently, AutoML emerged to provide mechanisms to automate parts of this process. However, most of the efforts focus on applying brute force procedures to try different algorithms or configuration and select the one which gives better results. To make a smarter and more efficient selection, a repository of knowledge is necessary. To this end, this paper proposes (1) an approach towards a common language to consolidate the current distributed knowledge sources related the algorithm selection in ML, and (2) a method to join the knowledge gathered through this language in a unified store that can be exploited later on, and (3) a traceability links maintenance. The preliminary evaluations of this approach allow to create a unified store collecting the knowledge of 13 different sources and to identify a bunch of research lines to conduct.

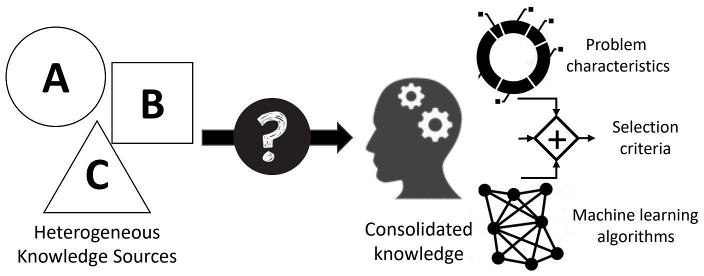

Figure 1 Problem motivation

Keywords: Machine Learning, Automated Machine Learning, Knowledge Representation, Model-Driven Engineering.

1 INTRODUCTION

Machine Learning (ML) entails the study of algorithms that automatically improve through experience [18]. This kind of algorithms has been successfully and broadly applied in the past [19] and nowadays is receiving increasing attention due to the affordable access to bigger computation power of machines.

A ML project requires selecting an appropriate algorithm to process the data to learn from, which is typically named *creating the data model*. However, there are thousands of algorithms under the paradigm of ML, each of them tailored to some specific tasks or contexts. In addition, many of these algorithms offer a different set of parameters to be configured (e.g., selecting the number of layers in a neural network).

Many existing approaches focus on the latter task, i.e., supporting the user after the algorithm selection is done, and few of them recommend an algorithm always after the user has provided the dataset. As an example, the recent research area of AutoML [28] aims to automate the different steps of ML projects. Nonetheless, such approaches neglect the early stages of the project. Many of them just provide a brute force mechanism that runs several algorithms in later stages of the project, i.e., when the dataset is ready. Thus, little effort has been done to support the user in the algorithm selection in an efficient manner (i.e., without applying brute force) and based on the problem characteristics (i.e., the early information).

The algorithm selection is specifically challenging since the existing knowledge regarding this task is distributed across different sources and each

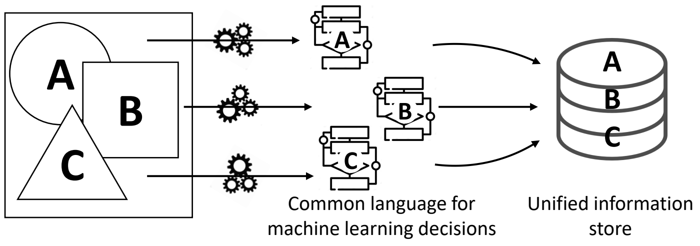
A
B
C
A
B
C
A
B
C
Common language for machine learning decisions
Unified information store

Figure 2 Overall proposal

of them is specified in a non-standard manner, thus, making it difficult to consolidate information from different sources, i.e., the name of the algorithms —or family of algorithms—, the selection criteria, and the characteristics of the problem that affect the selection are heterogeneous (cf. Figure 1).

To reduce the risk of taking inaccurate decisions due to a lack of information, a central repository of the ML Knowledge which stores the information in a structured way is required. In order to address this problem, this paper proposes (cf. Figure 2), on the one hand, a unified language for representing the knowledge related to the algorithm selection in ML projects. On the other hand, the paper describes a method to transform the knowledge gathered using this language to a unified knowledge store that can be exploited later on.

The unified language is presented as a metamodel that allows a graphical representation of the knowledge related to:

- The characteristics that affect the decisions, e.g., the amount of data that is available or the type of problem.
- The algorithms that can be recommended, e.g., Bayesian network or Support Vector Machine.
- The criteria for recommending an algorithm based on some of the characteristics, e.g., if the project aims to detect anomalies and the number of columns of the dataset is greater than 100, the Support Vector Machine algorithm is an excellent candidate.
- The source of knowledge from which recommendation criteria, algorithms, and characteristics are drawn, with their corresponding weighting, e.g. Scikit-learn with a weight of 90 percent.

Thereafter, the recommendations which are written using this language can be transferred to the proposed unified stored. This store allows reducing the ambiguity on (1) the name of the characteristics, e.g., the number of columns is often used as number of variables or features in some information sources, and (2) the name of the algorithms to recommend, e.g., Bayesian network can be found as Bayes model too.

This approach has been validated in a real industrial project using several publicly available information sources related to ML. Some of them represent the information as a picture summarizing the knowledge (e.g., [17,27]) while some others are expressed in text form (e.g., [7]). After considering an initial set of 13 different information sources, our approach successfully stored 150 recommendations of more than 80 algorithms considering more than 20 characteristics.

Although it remains out of the scope of this paper, this unified knowledge can be used to create a recommendation system that helps the user to decide which is the best algorithm for her new project.

This paper is an extension of a previous work [15] incorporating: (1) a new process, called term mappings, to improve the traceability between the original knowledge sources and the unified one, (2) the definition of a new concept called representative terms, (3) a new attribute to manage different versions of the knowledge source, (4) a new attribute to consider the weight of the knowledge sources according to their relevance, soundness, or reliability, to optimize the search for the right algorithm, and (5) a new motivating example that helps understanding the problem.

The rest of the paper is organized as follows: section 2 illustrates a motivating example on which the rest of the sections of this paper will be based, section 3 put in context the problem treated in this research, detailing a background related to the Machine Learning and Model-Driven Engineering. Section 4 presents the contributions of the paper, which is divided into three main elements: (1) a common language for recommendations of ML algorithms, (2) an early approach to reach a unified knowledge store, and (3) a plan to improve traceability of the term mappings. Finally, section 5 summarizes the conclusions and states a set of future work.

2 Motivating Example

With the aim of showing an example of the problems that this paper tries to solve, the following is an example of knowledge provided by two different sources, Microsoft Azure [17] and Dataiku [7]. Azure expresses the knowledge about machine learning in graph form with textual explanations. Unlike Azure, Dataiku expresses everything in text format.

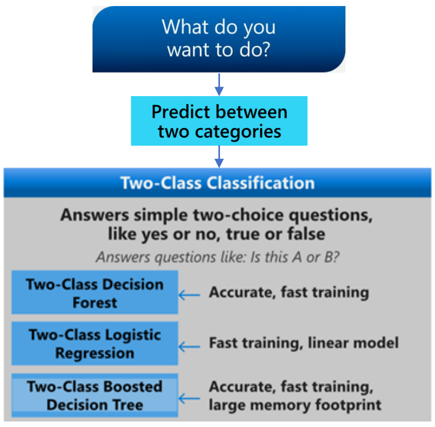
What do you want to do?
Predict between two categories
Two-Class Classification
Answers simple two-choice questions, like yes or no, true or false
Answers questions like: Is this A or B?
Two-Class Decision Forest
Accurate, fast training
Two-Class Logistic Regression
Fast training, linear model
Two-Class Boosted Decision Tree
Accurate, fast training, large memory footprint

Figure 3 Example knowledge source Microsoft Azure [17]

Table 1 Rule generation

Source	**Classes**	**Objective**	**Accuracy**	**Complexity**	**Algorithm**
Dataiku	yes/no	-	all possible outcomes	Simple	Decision Tree
Microsoft Azure	two	predict category	accurate	-	Decision Forest

This list shows an extract of Dataiku knowledge source. It contrasts with the example of Microsoft Azure, whose representation is much more visual:

- Logistic regression: The adaptation of a linear model to problems of classification. Based on an equation that spits out the prediction right after entering the variables. There is a tendency for the model to "overfit" and they tend to have trouble predicting more complex behaviors when the input variables are not independent.
- Decision tree: A series of yes/no rules based on the features, forming a tree, to match all possible outcomes of a decision. Not too powerful enough for complex data.

For instance, given a problem which is characterized by simple data, whose objective is to predict a category given by a binary variable, and considering the necessity of being precise, the above heterogeneous knowledge sources will trigger the rules showed in Table 1. To determine these rules, the characteristics referred to each of the sources have been split and then, compared with the value that this characteristic takes in the example problem.

This highlights the need to determine the equivalence between the terms of the different sources and whether they really refer to the same concept. Thus, it can be determined that "*yes/no*" and "*two-class*" represent the same concept and unify it in a single term. But it may not be trivial to determine whether "*Decision Tree*" and "*Decision Forest*" can be grouped in the same algorithm.

This example raises questions such as: Are they talking about the same algorithms? Are both sources equally relevant? Are the same characteristics taken into consideration? or, are they different?

Throughout this paper, it will be exposed a proposal of a standard language for knowledge representation and how to unify the knowledge of both sources in only one.

3 Background

This paper deals with two main concepts: ML (cf. Section 3.1) and Model-Driven Engineering (cf. Section 3.2).

3.1 Machine Learning

Machine learning emerges as a set of tools under a broader paradigm called artificial intelligence [18]. A ML project typically follows a life-cycle comprising many activities, e.g., the understanding of the problem, selecting the appropriate algorithm, parametrizing the algorithm, or creating and testing the model. All these steps involve tedious manual work which motivating the arising of AutoML [11,28], a research line that pursues the automation of the ML life-cycle steps.

So far, AutoML has been applied in several domains, like health [24], chemistry [9] or software engineering [1]. Most of the effort has been applied to automatically generate the ML model, e.g., to look for the algorithm's parameters which allow the most accurate model for a given dataset. However, the majority of these studies applied brute force to look for these parameters and only a few approaches [21, 24] include smarter solutions to reduce the search space.

In the industry field, there exist some commercial tools that support AutoML, e.g., BigML [4] or DataRobot [8]. Similarly to the academic field, these tools mainly apply brute force to find the appropriate algorithm.

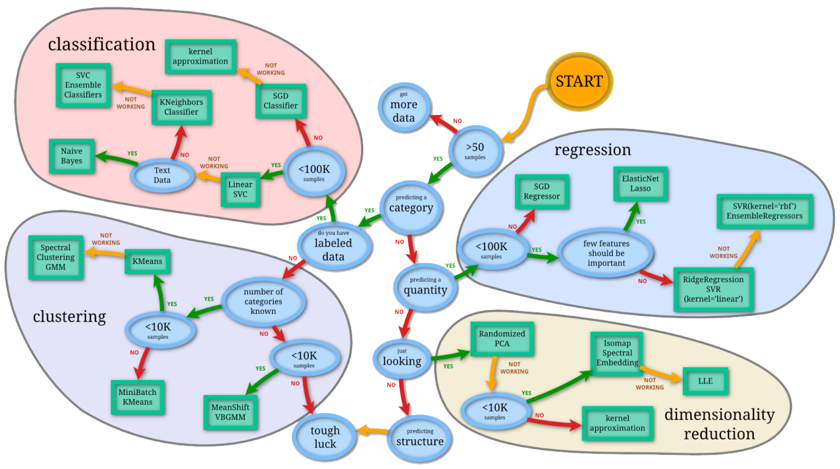

Figure 4 Example knowledge source. *source:scikit-learn.com*

Nonetheless, there are some tools such as TPOT [22] or H20 [14], which implement more sophisticated solutions, in which artificial intelligence techniques are used to choose the algorithm.

A common aspect that identifies these proposals is that they focus on finding the model with the best results. For this purpose, they only use their own experience, and do not give the possibility of including external knowledge within their solutions.

At a glance, selecting the appropriate algorithm is a non-deterministic and time-consuming task that depends on many problems and data characteristics. For this, the empirical knowledge is commonly shared in different Internet sources. Beyond the research papers, organizations used to share their experiences with the aim of guiding practitioners to use some software products.

Example 1 *Scikit-learn [25] (cf. Figure 4) and Microsoft Azure [17] share a cheat sheet which tries to explain which algorithm better fits according to a set of problem characteristics while Dataiku [7] contains technical documents with the same objective.*

The current work aims to gather all this distributed knowledge to enable a smarter way of AutoML. That is, modelling existing knowledge and reconciling it with the existing in the AutoML tool, building a unified knowledge that allows to automate more efficiently the choice of the algorithm.

3.2 Model-Based Engineering

The Model-Based Engineering (MBE) paradigm raises the use of models as a mechanism to reach the concrete from the abstract [12].

MBE incorporates the elements: concepts, notations, processes, rules and support tools [6], to provide advantages such as: having a common way of representing processes, facilitating compatibility with other formalisms, enabling the reuse of models or creating specific solutions of domain among others [20].

One of the main principles of this paradigm is that everything can be expressed as a model [3]. The term "metamodeling" is known as the action of modeling a model or modeling a modeling language. A metamodel is an abstraction of a model itself, which defines the properties of that model, the structure and restrictions for a family of models [16].

MBE is probably one of the best known paradigms in software engineering for modeling and it is considered a key component of the overall development process [5] .

This paradigm has numerous success stories in different domains such as: aeronautical [10] , cloud [2] , business process management [13] or software testing [29], among others.

Modeling languages are the mechanisms that allow designers to specify the models of their processes or systems. They establish the way in which the concrete representation of a conceptual model is defined and can be composed of graphical representations, textual representations, or even both. In any case, modeling languages are formally defined and oblige designers to use their syntax when creating models [6]. There are two major groups of modeling languages.

- Domain-Specific Languages (DSLs), which are designed specifically for a certain domain.
- General-Purpose Modeling Languages (GPMLs), which can be used for any application domain.

The Meta Object Facility (MOF) language [23], proposed by the reference body in this field, the Object Management Group (OMG), is one of the best-known languages for the definition of metamodels. In this language, metainformation is specified that makes data understandable by a computer [26].

Considering the background presented, it is possible to assume that MBE can be used to standardize the way in which the ML Knowledge is created.

4 Contribution

This section describes the main contributions of this paper. First, a common language is suggested to enable a consolidated way of representing the ML Knowledge (cf. Section 4.1), Since many decisions have to be made while joining the knowledge of different sources, a way to maintain the traceability related to the unification decisions is proposed (cf. Section 4.2). Finally, a process is described which incorporates such heterogeneous information sources into a unified knowledge store (cf. Definition 1) using the previous language (cf. Section 4.3).

Definition 1 *A* ***Unified knowledge store*** *UKS= (KnowlSources, AlgTerms, ReprAlgs, CharTerms, ReprChars, Rules) consists of*

- $KnowlSources$*: a set of tuples* $\langle source_{id}, source_{name}, source_{weight}\rangle$ *which contains a unique id in the* UKS*, the name of the knowledge source, and a weight associated to the confidence that this knowledge source holds.*
  Since the knowledge of each source might have different nature and be constructed using different methods (e.g., scientific knowledge tends to be less biased than opinion blogs), this weight is intended to subjectively asses the relevance, soundness, reliability or importance that should be given to the recommendations of a knowledge source. It consists of a percentage value which enables the comparison between sources.

  Example 2 *Let be a context where Dataiku recommends Logistic Regression and Microsoft recommends Decision Forest. In case that Microsoft was weighted with a higher value that Dataiku, it would mean that Decision Forest will be recommended before than Logistic Regression.*

- ***AlgTerms****: a set of tuples* $\langle alg_{id}, alg_{name}, source_{id}, reprAlg_{id}, \rangle$ *which contains a unique id in the* UKS*, the name of the algorithm which is given in the knowledge source, an id of a knowledge source in* $KnowlSources$ *which this algorithm comes from, and an id of a representative algorithm in* $ReprAlgs$*. The aim of the latter attribute is to keep a synonymous relationship.*
- ***ReprAlgs****: a set of tuples* $\langle reprAlg_{id}, reprAlg_{name}\rangle$ *which contains a unique id in the* UKS *and a name of the representative algorithm, i.e., a categorical representative of similar algorithms.*

Example 3 *The algorithm in $AlgTerms$ related to the "Boosted Decision Tree", from Azure, and "Decision Tree", from Dataiku, can share a common representative in $ReprAlgs$ with a name "D.Tree".*

- ***CharTerms**: similarly to $AlgTerms$, is a set of tuples $\langle char_{id}, char_{name}, source_{id}, reprChar_{id}, \rangle$ which contains a unique id in the UKS, the name of the characteristic, an id of a knowledge source in $KnowlSources$ which it comes from, and an id of a representative characteristic in $ReprChars$.*
- ***ReprChars**: similarly to $ReprAlgs$, is a set of tuples $\langle reprChar_{id}, reprChar_{name} \rangle$ which contains a unique id in the UKS and a name of the representative characteristic.*
- ***Rules**: a set of tuples $\langle rule_{id}, antecedents, consequences, source_{id} \rangle$ which contains a unique id in the UKS, the reference of the knowledge source which describes this rule, and the rule itself, i.e., the antecedents which fire the consequences. On the one hand, $antecedents$ is a set of pairs $\langle reprChar_{id}, value \rangle$ stating that this rule is fired if the characteristics have the given values. On the other hand, $consequences$ is a set $reprAlg_{id}$ which indicates that these algorithms are recommended if the rule is fired.*

4.1 A common language for recommendations of ML algorithms

This paper proposes a formal language to abstract from the different languages which are used to represent the knowledge. More precisely, in the context of recommendations for the usage of a ML algorithm, sources of knowledge can be found in research papers, Web forums, cheat sheets of organizations, etc. Referring to the same algorithms and characteristics with different terms. Nonetheless, if these sources are analyzed and the non-relevant information is wiped out, the knowledge that they contain shares a similar and simple format: some algorithms are recommended if a set of problem characteristics have some specific values (e.g., the problem is to $predict$ between $two\ categories$, and the result must be $accurate$ so it is recommended $logistic\ regression$).

Herein, we propose an abstract syntax or metamodel (cf. Figure 5) that allows: (1) standardizing the way in which knowledge is gathered and (2) being interpreted by a computer program.

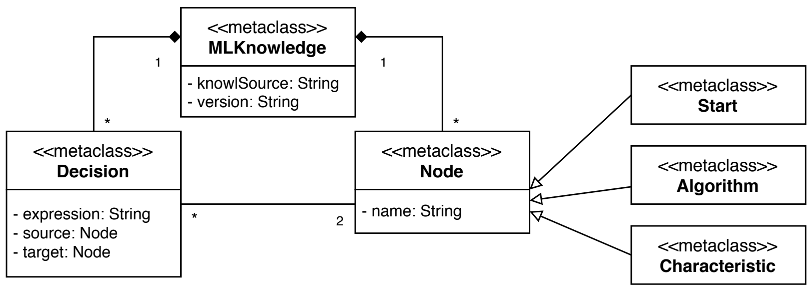

Figure 5 MLKnowledge Metamodel

The objective of this modeling is to allow the representation of the knowledge of any single source in a common language [1].

The proposed metamodel is composed of six metaclasses.

The "*MLKnowledge*" metaclass, defined by the attribute "*knowlSource*" and "*version*", allows the content of the knowledge source to be represented in the format of the target knowledge source and to have a record of their updates. This representation format is composed of Decision and Nodes.

The "*Node*" metaclass, defined by the attribute "*name*", allows to represent each origin points of the different branches of the knowledge source. This metaclass can be represented as three different ways:

- **Start**: it represents the initial node.
- **Characteristic**: it represents the antecedents that are considered for making a decision i.e. generating a consequence.
- **Algorithm**: it represents a consequence, i.e., an algorithm resulting from the recommendation based on some antecedents.

The "*Decision*" metaclass allows to represent the antecedents of the knowledge source, i.e., a set of characteristics that affects the decisions and the criteria for recommending an algorithm based on them. These criteria are represented through the "expression" attribute of "*Decision*" metaclass. Moreover, this metaclass connects instances of the metaclass "*Node*" through the "*source*" and "*target*" attributes.

In addition to the abstract syntax, a concrete syntax that allows to create models based on the ML Knowledge Language was defined. This concrete syntax is a DSL composed of a set of specific symbols (cf. Figure 6) that let the software engineer instantiate each of the metaclasses of the metamodel.

[1] Note that the metamodel which is suggested in this paper is not intended to represent sets of knowledge source but a single one.

Metaclass	**Description**	**Symbol**
Start Node	It represents the initial node	**Start**
Characteristic Node	It represents a feature of the knowledge source	◇
Algorithm Node	It represents an algorithm	*name*
Decision	It represents the decision based on certain criteria and characteristics	$\xrightarrow{\textit{expression}}$

Figure 6 Concrete syntax definition

A small example of the use of the DSL is illustrated in Figure 7. This figure is based on an extract of the Microsoft's knowledge source (cf. Figure 3) which is modeled with the DSL described above.

As evidenced in Figure 7, the model begins with "*What do you want to do?*" so it is identified as the initial node (i.e., *Start Node*). Next, the "*Predict between two categories*" node filters by two characteristics, the *Prediction objective* and the *Number of categories*. Thereafter, two nodes are obtained in the DSL, one for each characteristic filter. For the sake of simplicity, Figure 3 is just an extract from the knowledge source and, thus, the complete graph cannot be represented. Therefor, from *Prediction objective* two decisions come out, *Values* and *Categories*. But Figure 7 only connects a new node, *Number of categories*, by the *Category* decision, leaving open the other branch.

For modeling the rest of the extract, the relevance of the characteristics which are associated with each algorithm are taken according to their order, i.e., the first characteristic is the most important and so on.

Example 4 *For example, this part of Figure 3 is interpreted in the following way: Two-Class Decision Forest is the most accurate and Two-Class Logistic regression is the fastest. In addition, three other characteristics are extracted: training speed, accuracy importance and whether it has a large memory capacity.*

First, Training speed corresponds to a characteristic that is the decision target of the previous node, i.e., Number of categories with the expression 2*. The characteristic Training speed is connected by a decision with the expression $Fast$, to the Accuracy importance characteristic.*

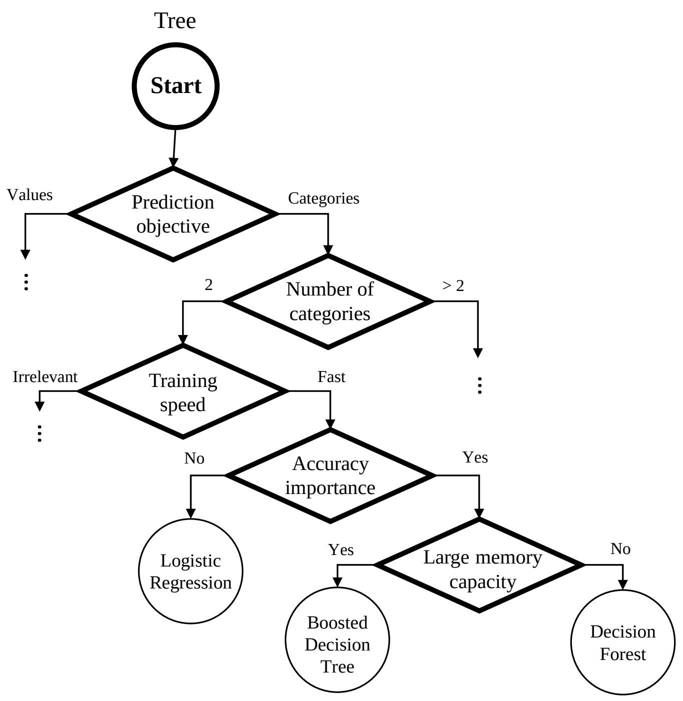

Figure 7 Example of DSL use

Second, Accuracy importance characteristic is connected to a two "Nodes": the Two-Class Logistic regression algorithm —if Accuracy importance takes the expression Yes—, and the "Large memory capacity" characteristic —if Accuracy importance takes the expression Not—.

Finally, "Large memory capacity" is connected to a pair of algorithm nodes called "Two-Class Boosted Decision Tree" and "Two-Class Decision Forest" by Yes or No respectively. It means, if the decision source expression "large memory capacity" takes value yes, the consequence value will be "Two-Class Boosted Decision Tree", otherwise, the consequence value will be "Two-Class Decision Forest".

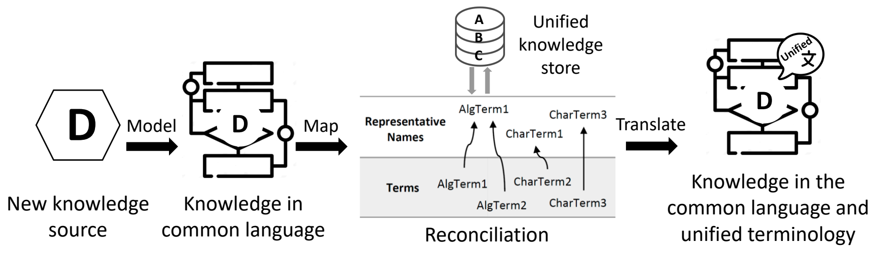

Figure 8 Consolidate terms of knowledge source

4.2 Terms mappings

In a general view, as seen in Figure 2, this approach is divided into two major stages. This section describes the first one, where each knowledge source is expressed using the suggested language (cf. Figure 8), i.e., the knowledge source is manually modeled using the previous language and, thus, the different relations between the problem characteristics and the recommended algorithms are written in a formal way. In order to deal with the ambiguity and similarities between knowledge sources, this stage includes a $reconcilitation$ phase where the terms of a knowledge source (i.e., characteristics and algorithms) are mapped to terms in the UKS, i.e., the terminology of the considered knowledge source is unified with the terminology that is already used.

First, the UKS is updated by including the new $KnowlSource$ with a new $source_{id}$ including the $weight$ which is given for the recommendations of this source (cf. Definition 1). Second, the characteristics and algorithms are included in $CharTerms$ and $AlgTerms$ respectively. In case that no synonym can be found in the UKS for the new characteristic or algorithm, a new categorical representative is also included in the $ReprChars$ or $ReprAlg$ respectively. In addition, the new term is associated with the latter representative. However, in case that a term already exists with equivalent meaning, the new term is associated with the same categorical representative as the equivalent one has.

Finally, the model of the knowledge source is updated by substituting each term by the categorical representative which has been associated in the UKS, thus resulting in a knowledge source which is expressed in the common language with a unified terminology.

Example 5 *Figure 6 uses the common language to represent the knowledge which is extracted from the Microsoft Azure knowledge source. Herein, the* *nodes Prediction objective, Number of categories, Training speed, Accuracy importance, Logistic Regression, Boosted Decision Tree, Large memory capacity and Decision Forest are found (ignoring the $Start$, which is always included). These nodes of algorithms and characteristics are automatically included as terms (i.e., $AlgorithmTerms$ and $CharacteristicTerms$ respectively) in the UKS, and associated with its corresponding $KnowlSource$, in this case, $Microsoft$. After this, a categorical representative must be associated to each new term. Considering that the term Large memory capacity has not a similar term in the UKS, it is included as a $CharTerm$ and as a $ReprChar$ node. However, in the case of Accuracy importance, assuming that there is already a $ReprChar$ with the name Importance of precision, the Accuracy importance term is included and associated with this $ReprChar$ node, since they have an equivalent meaning.*

Term $reconciliation$ is a crucial task for both (1) getting value from the aggregated knowledge in the UKS, and (2) maintaining a traceability mechanism to reduce the impact of future changes. On the one hand, the knowledge sources are not only associated with the terms, but also to the decisions. So, given a decision between two nodes, it is possible to determine the knowledge source from which it comes, and therefore the weight that is given to that decision (i.e., the same weight of the knowledge source). On the other hand, in case that a change has to be made in the terms of the UKS (e.g., divide a general term in two more concrete ones), tracing the impact of the knowledge source that has to be revised is straight forward since the source is stored on each synonym and they are linked.

4.3 Towards a unified knowledge store

This section described the second stage of the proposal where the knowledge source is processed to extract the individual recommendation rules and store them in the unified knowledge store (cf. Figure 9). For this, since the model present a tree-like structure, it is divided into the different paths that exist from the root (i.e., the start node) to any algorithm node.

Each path is composed of (1) a "*Start Node*, (2) a set of "*Characteristic Nodes* together with a labeled outgoing edge where the label indicates the value that takes the characteristic, and (3) one or various "*Algorithm Nodes*. Therefore, the paths are processed to extract the rules of the knowledge source. These rules have a similar shape to the unified knowledge store. Each one keeps a set of antecedents (i.e., the names and the values of the

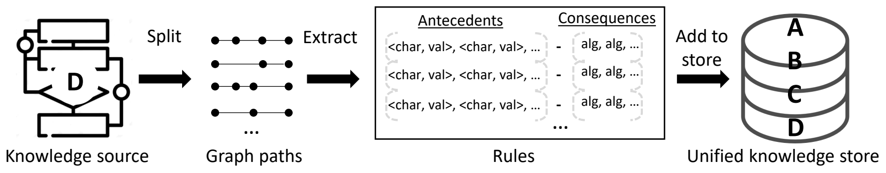

Figure 9 Add knowledge to the unified knowledge store

characteristics that appear in the path) and a set of consequences (i.e., the names of the algorithms that appear in the path).

For this, each characteristic name (and algorithm name) that exists in the antecedents (and consequences) are substituted by the $reprChar_{id}$ (and the $reprAlg_{id}$). Finally, these intermediate rules are incorporated in the unified knowledge store. That is, a new $rule_{id}$ is obtained and the tuple $\langle rule_{id}, antecedents, consequences, source_{id}\rangle$ is stored in the $Rules$ set of the unified knowledge store.

For example, Figure 3 depicts how the different rules can be extracted from the model of Figure 7.

The algorithm selection process will change each time that a new knowledge source is included in the UKS, as a new source generates new knowledge into the UKS, i.e., new rules and new possibly representative terms or modifications to existing ones. In this sense, the use of the UKS is based on a comparison of the characteristics of the ML problem with the representative characteristics of the UKS. This results in the recommendation of representative algorithms which are obtained from the rules that meet the characteristics of the ML problem. It is important to emphasize that the UKS knowledge undergoes a regular update process to check for new content from each source. Therefore, each content update will be reflected in the version of the knowledge source. In this way, the end user of this proposal will always be using the latest updated version of the UKS.

5 Conclusions and Future Work

This paper presents an approach to deal with the distributed knowledge of ML. Specifically, it aims to create a repository with rules that help to decide which ML algorithms are suitable to solve a given problem. For this, a common language for modeling this knowledge is proposed. Such a language is stated in form of a metamodel that a computer program can process. In addition, a procedure to transfer these models to a unified knowledge store is described. This store will enable exploiting the knowledge of the distributed sources to make decisions with less risk and will make it possible to maintain the traceability of terms mappings to their knowledge source. So that relationships between terms and representative terms can be changed if an inconsistency is subsequently detected. Furthermore, each knowledge source is weighted, which will allow establishing differences between the recommendations according to their origin.

This results in a unified knowledge store, which allows an external user to obtain an algorithm recommendation depending on the specific characteristics of her ML problem. For this, she compares the representative characteristics with those of her problem and, according to its values, some of the rules are fired with a weight associated to the knowledge source. Although all these rules may recommend an algorithm, only the one that takes greater weight is selected.

However, this work considers some assumptions that limit its application. First, the suggested unified knowledge store keeps simple rules lacking more complex syntax like OR or NOT expressions. Second, it considers that knowledge sources do not evolve, so no versioning or dating of terms is stored. And finally, using this proposal requires manual work to model the knowledge source through the provided language which may entail a considerable effort depending on the number of sources. Nonetheless, this effort will be leveraged not only by a single ML project but by the future ones too.

As further future work, we plan (1) to exploit the unified knowledge store in order to generate a decision support tool, (2) to extend the traceability scope to keep track of every decision which is made from the early stage of translating the knowledge source into the common language, (3) to extend the proposed MLKnowledge language capabilities since complex expressions, such as disjunctions and negations, occasionally appear within the antecedents of knowledge sources, and (4) to introduce the concept of intensity of recommendation with the aim of expressing the degree of acceptance of the recommendations, since some knowledge sources express a distinction between the value of different recommendations (e.g., excellent vs acceptable recommendations)

ACKNOWLEDGEMENTS

This research has been supported by the Pololas project (TIN2016-76956-C3-2-R) of the Spanish Ministry of Economy and Competitiveness, and the PROMETEO (P009-18/E09) project of the Centro para el Desarrollo Tecnológico Industrial (CDTI) of Spain.

References

[1] Hadil Abukwaik, Andreas Burger, Berima Kweku Andam, and Thorsten Berger. Semi-automated feature traceability with embedded annotations. *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 529–533, 2018.

[2] Danilo Ardagna, Elisabetta Di Nitto, Parastoo Mohagheghi, Sébastien Mosser, Cyril Ballagny, Francesco D'Andria, Giuliano Casale, Peter Matthews, Cosmin-Septimiu Nechifor, Dana Petcu, et al. Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds. In *2012 4th International Workshop on Modeling in Software Engineering (MISE)*, pages 50–56. IEEE, 2012.

[3] Jean Bézivin. On the unification power of models. *Software & Systems Modeling*, 4(2):171–188, 2005.

[4] BigML. Bigml. Available at: `https://bigml.com/`, 2019. Last accessed: July 2019.

[5] Francis Bordeleau and Edgard Fiallos. Model-based engineering: A new era based on papyrus and open source tooling. In *OSS4MDE@ MoDELS*, pages 2–8. Citeseer, 2014.

[6] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 1(1):1–182, 2012.

[7] Dataiku. Dataiku blog. Available at: `https://blog.dataiku.com/`, 2019. Last accessed: July 2019.

[8] DataRobot. Datarobot. Available at: `https://www.datarobot.com/`, 2019. Last accessed: July 2019.

[9] Steven L. Dixon, Jianxin Duan, E. Drybrough Smith, Christopher D Von Bargen, Woody Sherman, and Matthew P. Repasky. Autoqsar: an automated machine learning tool for best-practice quantitative structure-activity relationship modeling. *Future medicinal chemistry*, 8 15:1825–1839, 2016.

[10] María José Escalona, Julián Alberto García-García, Fernando Mas, Manuel Oliva, and Carmelo Del Valle. Applying model-driven paradigm: Calipsoneo experience. In *CAiSE Industrial Track*, pages 25–32. Citeseer, 2013.

[11] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015.

[12] Frédéric Fondement and Raul Silaghi. Defining model driven engineering processes. In *Third International Workshop in Software Model Engineering (WiSME), held at the 7th International Conference on the Unified Modeling Language (UML)*, 2004.

[13] Julián Alberto García-García, Laura García-Borgoñón, María José Escalona, and Manuel Mejías. A model-based solution for process modeling in practice environments: Plm4bs. *Journal of Software: Evolution and Process*, 30(12):e1982, 2018.

[14] H20. Automl: Automatic machine learning. Available at: `http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html`, 2019. Last accessed: September 2019.

[15] Antonio Martínez-Rojas., Andrés Jiménez-Ramírez., and Jose G. Enríquez. Towards a unified model representation of machine learning knowledge. In *Proceedings of the 15th International Conference on Web Information Systems and Technologies - Volume 1: APMDWE,*, pages 470–476. INSTICC, SciTePress, 2019.

[16] Stephen J Mellor, Kendall Scott, Axel Uhl, and Dirk Weise. *MDA distilled: principles of model-driven architecture*. Addison-Wesley Professional, 2004.

[17] Microsoft. Machine learning algorithm cheat sheet for azure machine learning studio. Available at: `https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-cheat-sheet`, 2019. Last accessed: July 2019.

[18] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[19] Tom Michael Mitchell. *The discipline of machine learning*, volume 9. Carnegie Mellon University, School of Computer Science, Machine Learning . . . , 2006.

[20] Parastoo Mohagheghi, Wasif Gilani, Alin Stefanescu, and Miguel A Fernandez. An empirical study of the state of the practice and acceptance of model-driven engineering in four industrial cases. *Empirical Software Engineering*, 18(1):89–116, 2013.

[21] Felix Mohr, Marcel Wever, and Eyke Hüllermeier. Ml-plan: Automated machine learning via hierarchical planning. *Machine Learning*, 107(8):1495–1515, Sep 2018.

[22] Randal S Olson, Nathan Bartley, Ryan J Urbanowicz, and Jason H Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 485–492, 2016.

[23] OMG. Meta object facility (MOF) 2.5 core specification. Version 2.5.1. Available at: `https://www.omg.org/spec/MOF/2.5.1/PDF`, 2016. Last accessed: July 2019.

[24] Maria S. Panagopoulou, Makrina Karaglani, Ioanna Balgkouranidou, Eirini Biziota, Triantafillia Koukaki, Evaggelos Karamitrousis, Evangelia Nena, Ioannis Tsamardinos, George Kolios, Evi S Lianidou, Stylianos Souglakos John Kakolyris, and Ekaterini Chatzaki. Circulating cell-free dna in breast cancer: size profiling, levels, and methylation patterns lead to prognostic and predictive classifiers. *Oncogene*, 38:3387–3401, 2018.

[25] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[26] Douglas C Schmidt. Model-driven engineering. *COMPUTER-IEEE COMPUTER SOCIETY-*, 39(2):25, 2006.

[27] Sckit-learn. sckikit-learn algorithm cheat-sheet. Available at: `https://scikit-learn.org/stable/tutorial/machine\_learning\_map/index.html`, 2019. Last accessed: July 2019.

[28] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-weka: Automated selection and hyper-parameter optimization of classification algorithms. *CoRR*, abs/1208.3719, 2012.

[29] Emília Villani, Rodrigo Pastl Pontes, Guilherme Kisselofl Coracini, and Ana Maria Ambrósio. Integrating model checking and model based testing for industrial software development. *Computers in Industry*, 104:88–102, 2019.