ARCHIVIO ISTITUZIONALE
DELLA RICERCA

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

HAMLET: A framework for Human-centered AutoML via Structured Argumentation

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

Published Version:
Francia M., Giovanelli J., Pisano G. (2023). HAMLET: A framework for Human-centered AutoML via Structured Argumentation. FUTURE GENERATION COMPUTER SYSTEMS, 142, 182-194 [10.1016/j.future.2022.12.035].

Availability:
This version is available at: https://hdl.handle.net/11585/913288 since: 2023-02-01

Published:
DOI: http://doi.org/10.1016/j.future.2022.12.035

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/).
When citing, please refer to the published version.

(Article begins on next page)

16 August 2024

HAMLET: a framework for Human-centered AutoML via Structured Argumentation

Matteo Francia[a], Joseph Giovanelli[a,*], Giuseppe Pisano[b]

[a]*DISI – University of Bologna, Italy*
[b]*Alma AI – University of Bologna, Italy*

Abstract

Machine Learning (ML) plays a crucial role in data analysis and data platforms (i.e., integrated sets of technologies that collectively meet end-to-end data needs). In the last decade, we have witnessed an exponential growth in both the complexity and the number of ML techniques; leveraging such techniques to solve real-case problems has become difficult for Data Scientists. Automated Machine Learning (AutoML) tools have been devised to alleviate this task, but easily became as complex as the ML techniques themselves—with Data Scientists losing again control over the process. In this paper, we design and extend HAMLET (Human-centered AutoMl via Logic and argumEnTation), a framework that helps Data Scientists to redeem their centrality. HAMLET enables Data Scientists to express ML constraints in a uniformed human- and machine-readable medium. User-defined constraints are interpreted to drive the exploration of ML pipelines (i.e., Data Pre-processing transformations shape the data so that the ML task can be performed at its best). AutoML retrieves the most performing pipeline instance, and finally, new constraints are learned and integrated through Logic and Argumentation. By doing so, HAMLET not only allows an easy exploitation of the knowledge acquired at each iteration, but also enables its continuous revision via the AutoML tool and the collaboration of both Data Scientists and domain experts.

Keywords: Human-centered, AutoML, Logic, Argumentation, CRISP-DM, Data Scientist

1. Introduction

Data platforms, integrated sets of technologies that collectively meet end-to-end data needs, work towards the automation of data management and analysis [1]. Machine Learning (ML) plays a key role in such processes (e.g., to devise cost models for querying data over heterogeneous data sources [2] and manage data through lineage [3]; many applications are well surveyed in [4]). Data platforms aim at supporting end-to-end data analysis; in this scope, the Cross-Industry Standard Process for Data Mining (CRISP-DM) [5] is the most acknowledged standard process model—and we will take it as a reference model hencefort. Given a Machine Learning task to solve, the Data Scientist (DS) collects raw data in arbitrary formats (e.g., from the data lake), builds up knowledge on both the problem and the data, translates such knowledge into *constraints*, designs and trains a model, and finally deploys the solution as a new component integrated into the data platform. Such a solution consists of a *ML pipeline*: a sequence of *Data Pre-processing transformations* ending with an *ML task*. The DS instantiates the pipeline among a large set of *algorithms*, which, in turn, can potentially have many *hyperparameters*. The accuracy of the deployed solution depends on finding both the best algorithms along with their hyperparameters within an exponential search space.

Automated Machine Learning (AutoML) tools assist the DS in finding such an ML pipeline. They leverage state-of-the-art optimization approaches to smartly explore huge search spaces of solutions. AutoML has been demonstrated to provide accurate performance, even in a limited time/iteration budget. When setting up the search space, it is highly important for the DS to inject her knowledge about the problem into constraints that prevent the AutoML tool to retrieve invalid solutions (i.e., the result of those cannot be deemed correct). However, the support to constraint/knowledge injection is limited and the AutoML tools became that complex to make it difficult for the DS to understand their functioning, hence losing control over the process [6].

The need for a Human-centered and explainable framework for AutoML is real [7, 8, 9] (or even mandatory in recent analytic scenarios where the user is interacting with mixed-reality and smart assistants [10, 11]). It is crucial for the DS to augment her knowledge by learning new insights (e.g., new constraints) from the retrieved solutions. Indeed, the DS requires understanding the AutoML process in order to trust the proposed solutions [12]. Some works [7, 8, 9] prescribe the usage of a Human-centered framework for AutoML, yet they only suggest design requirements. Alternatively, the authors in [13] have proposed a tool that visualizes the best and the worst solutions retrieved by an AutoML tool. We claim that a Human-centered framework should provide the mechanisms to: (i) help the DS to structure her knowledge about the problem in an effective search space; and (ii) augment the knowledge initially possessed by the DS with the one produced by the AutoML

*Corresponding author
Email addresses: m.francia@unibo.it (Matteo Francia), j.giovanelli@unibo.it (Joseph Giovanelli), g.pisano@unibo.it (Giuseppe Pisano)

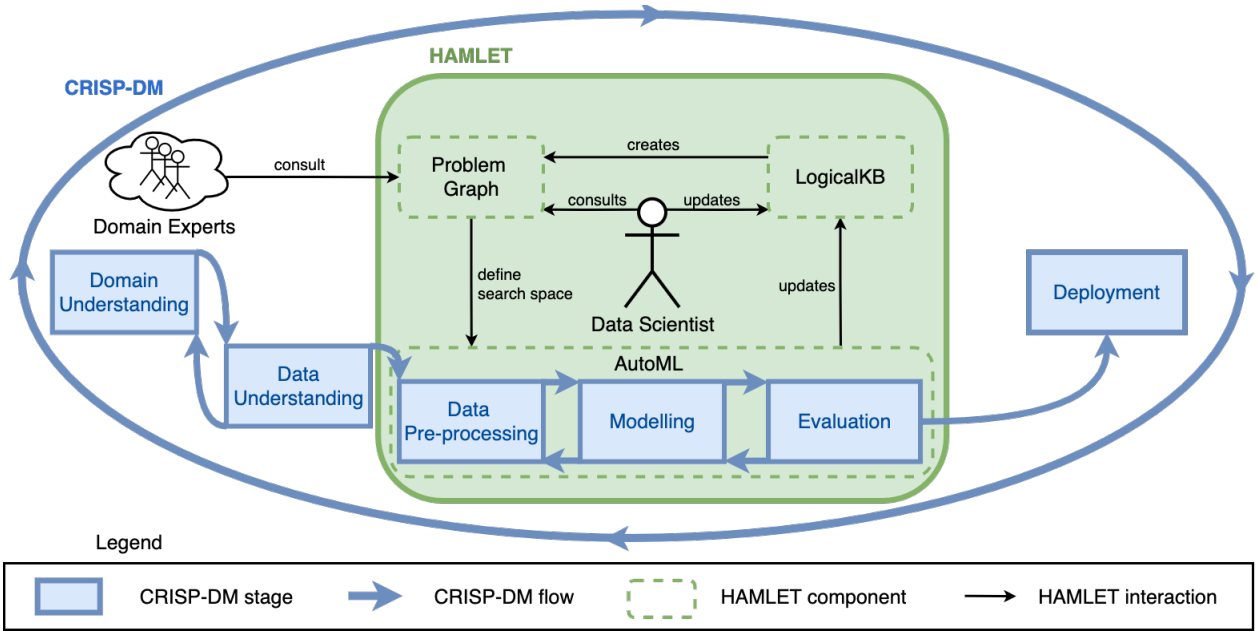


Figure 1: Integrating HAMLET with the CRISP-DM process model.

optimization process.

For this purpose, we introduce HAMLET (Human-centered AutoMl via Logic and argumEnTation; Figure 1), a framework that enhances AutoML with Structured Argumentation to: structure the constraints and the AutoML solutions in a Logical Knowledge Base (LogicalKB); parse the LogicalKB into a human- and machine-readable medium called Problem Graph; devise the AutoML search space from the Problem Graph; and leverage the Problem Graph to allow both the DS and an AutoML tool to revise the current knowledge. This paper extends and engineers the vision proposed in [14] as follows.

(i) We provide an innovative formalization of the AutoML problem, which considers ML pipelines of multiple lengths, Data Pre-processing steps and user-defined constraints.
(ii) We design the formal foundation of HAMLET, supporting the injection of constraints to select ML pipelines as well as the resolution of possible arising inconsistencies.
(iii) We implement a functioning prototype of HAMLET.
(iv) We provide a preliminary empirical evaluation, including the overhead introduced by the argumentation process and the comparison against state-of-the-art algorithms.

The remainder of the paper is structured as follows. In Section 2, we introduce the related work and necessary background. Then, we provide the problem formulation in Section 3 and its implementation in Section 4. Finally, we provide some preliminary evaluation in Section 5, and we draw the conclusions and future research directions in Section 6.

2. Background and Related Work

HAMLET intersects two research areas, *Automated Machine Learning* and *Argumentation*. To the best of our knowledge, no contribution lies in this intersection to provide a Human-centered approach for the optimization of ML pipelines.

2.1. Automated Machine Learning

AutoML tools lighten the DS in the overwhelming practice of finding the best ML pipeline instance (AutoML contributions mainly refer to supervised tasks). In the early days, only the optimization of the ML task was addressed (but no pre-preprocessing). Auto-Weka [15] formalized the problem as "combined algorithm selection and hyperparameter optimization": various ML algorithms and hyperparameters are tested over a dataset to find the most performing configuration. Such optimization was successfully implemented by leveraging Bayesian optimization [16], a sequential strategy for global optimization: until a limit (budget) of iterations or time is reached, an increasingly accurate model is built on top of the previously explored configurations.

Recently, AutoML is no longer limited to optimizing just the ML task, but it also includes Data Pre-processing [17, 18]. In doing so, Auto-sklearn [19] fixes the arrangement of the transformations a priori, without considering that the most performing arrangement changes according to the problem and dataset at hand. However, considering several arrangements translates into larger search spaces that are not easy to explore.

Several improvements have been made to let AutoML tools explore as many configurations as possible. Multi-fidelity methods [20] (i.e., the use of several partial estimations to boost the time-consuming evaluation process) have been exploited. Meta-learning leverages the previous performance of pipeline instances on a wide range of different datasets to provide several recommendations for the dataset at hand, such as promising pipeline instances (possibly acting as an alternative to Bayesian optimization) and search spaces producing good

performance. Yet, meta-learning per se performs poorly, because it provides coarse-grained recommendations, while it is beneficial in warm-starting Bayesian optimization (i.e., the suggested pipeline instances are visited at the beginning to boost the convergence process). With respect to HAMLET, meta-learning per se is not expressive enough to be considered as an alternative, but can be leveraged as a support in building the LogicalKB (i.e., learning constraints that resulted effective on many similar datasets).

We believe that the DS has the duty to revise and supervise the suggested solutions as well as the process producing them. Yet, stacking (more and more) complex mechanisms on top of each other unavoidably led to a less understandable optimization that can be hardly controlled by the DS (especially if without a strong computer science background).

2.2. Towards Human-centered AutoML Approaches

As of now, the DS role in AutoML is limited to choosing the dataset to analyze, the validation technique (e.g., cross validation, hold out), and the metric to optimize (e.g., accuracy, F1 score). AutoML researchers aim at making ML accessible to a wider audience; this has been addressed first by improving automation and now by improving transparency, which also enables human intervention when needed. Auto-Weka [15] and Auto-Sklearn [19] enables non-expert users to build ML models, but the "black-box" can be barely open. Indeed, as advocated in [12], DSs require to understand the process to trust the proposed solutions. This direction, named "Human-centered AutoML", is pursued by both researchers and companies.

As to research contributions, we found plenty of visualization wrappers. In [12], the authors raise the need of incorporating transparency into AutoML: after a session interview, they discover that – out of all their proposed features – model performance metrics and visualizations are the most important information to DSs when establishing their trust in the proposed solutions. ATMSeer [21] provides different multi-granularity visualizations to enable users to monitor the AutoML process and analyze the searched models. PipelineProfiler [13] offers interactive visualizations of the AutoML outputs and enables the reproducibility of the results through a Jupiter notebook. Other contributions enhance current AutoML techniques towards easier human-interactions by: (i) supporting ethic and fair constraints in Bayesian Optimization through a mathematical encoding [22, 23]; (ii) simplifying the usage of AutoML with symbolic annotations [24] and declarative languages [25]; (iii) supporting fast feed-backs from AutoML (i.e., runs that are less time-consuming) by leveraging well-known mechanisms of the DBMS (e.g., lineage optimization) [26, 27]. Recently, MILE [8] has proposed to perform AutoML analysis with an end-to-end framework that reflect a DBMS (i.e., a query language + a lineage optimization).

Companies like Google and IBM are the ones most engaged in boosting the involvement of the human in the loop. Google Vizer [28] and Google Facets[1] are the two main visualization tools. The former reveals details of the different hyperparameters tried in the optimization [28], and the latter focuses on analyzing the output and recognizes biased AI (e.g., ML models that discriminate on sensible attributes such as gender). As to IBM, AutoAI [29] and AutoDS [30] are the tools developed within the MIT-IBM Watson AI Lab. Specifically, the former enables non-technical users to define and customize their business goals as constraints. The latter assists the DS team throughout the CRISP-DM process (e.g., in data collection and pipeline design [31, 30] and in the augmentation of the DS's knowledge about the dataset features [12]).

Overall, several studies have been made to understand the proper design of a Human-centered AutoML tool. In [32], the authors overview the main AutoML issues; while in [33] authors suggest improvements towards the Human-centered shift. In [7, 6, 34], interviews with DSs are conducted to reveal their perception of AutoML as well as their needs and expectations in the next-generation tools. The main insight is that the future of data science work will be a collaboration between humans and AI systems, in which both automation and human expertise are indispensable [9]. To this end, AutoML should focus on: simplicity, reproducibility, and reliability [6, 34].

While the above-mentioned papers mainly focus on visualization, HAMLET brings the DS in the loop by allowing her to inject knowledge in the form of constraints, optimizing and learning new constraints through AutoML, and managing such constraints and conflicts through Argumentation.

2.3. Logic and Argumentation

Logic is defined as the abstract study of statements, sentences and deductive arguments [35]. From its birth, it has been developed and improved widely, now including a variety of formalisms and technologies.

Argumentation is a well-known formal tool for handling conflicting information (e.g., opinions and empirical data). In Abstract Argumentation [36], a scenario (e.g., a legal case) can be represented by a directed graph. Each node represents an argument, and each edge denotes an attack by one argument on another. Each argument is regarded as atomic. There is no internal structure to an argument. Also, there is no specification of what is an argument or an attack. A graph can then be analyzed to determine which arguments are acceptable according to some general criteria (i.e., semantics) [37].

A way to link Abstract Argumentation and logical formalisms has been advanced in the field of Structured Argumentation [38], where we assume a formal logical language for representing knowledge (i.e., a LogicalKB) and for specifying how arguments and conflicts (i.e., attacks) can be derived from that knowledge. In the structured approach, the premises and claims of the argument are made explicit, and the relationship between them is formally defined through rules internal to the formalism. We can build the notion of attack as a binary relation over structured arguments that denotes when one argument is in conflict with another (e.g., contradictory claims or premises). One of the main frameworks for Structured Argumentation is ASPIC$^+$[39]. In this formalism, arguments are built with two

[1] `https://pair-code.github.io/facets/`

kinds of inference rules: strict rules, whose premises guarantee their conclusion, and defeasible rules, whose premises only create a presumption in favor of their conclusion. Then conflicts between arguments can arise from both inconsistencies in the LogicalKB and the defeasibility of the reasoning steps in an argument (i.e., a defeasible rule used in reaching a certain conclusion from a set of premises can also be attacked).

Once defined the right logical language for encoding the DS and AutoML knowledge, a Structured Argumentation model (e.g., an ASPIC$^+$ instance [40]) can support HAMLET with the formal machinery to build an Argumentation framework upon the data, while Abstract Argumentation would dispense the evaluation tools.

3. Problem Formulation

Figure 1 illustrates the overview of HAMLET. When addressing end-to-end data analysis, a DS usually follows a process model such as CRISP-DM. The DS starts by collecting raw data in an arbitrary format. Then, "Domain Understanding" is conducted. The DS works in close cooperation with domain experts and enlists *domain-related constraints* (i.e., intrinsic of the problem). Follows "Data Understanding", devoted to data analysis, and to extract *data-related constraints* (e.g., defined by the data format). Domain and Data Understanding might be repeated many times until the DS is satisfied by the acquired knowledge. Once confident, the DS investigates different solutions throughout "Data Pre-processing", "Modelling", and "Evaluation". Data Pre-processing and Modelling are conducted to effectively build the solution, while Evaluation offers a way to measure its performance. Such a solution consists of a *ML pipeline*: a sequence of *Data Pre-processing transformations* ending with an *ML task*. The DS instantiates different pipelines among a large set of algorithms; the performance are affected by both the algorithms and some exposed hyperparameters. While seeking the best performing and valid solution, the DS should consider the already known constraints – domain- and data-related – and the ones she discovers during Data Preprocessing and Modelling, respectively: *transformation-* and *algorithm-related constraints* (e.g., due to the intrinsic semantic of transformations and algorithms at hand). Finally, the process concludes with the "Deployment" of the solution.

HAMLET intersects CRISP-DM, allowing the DS to inject and augment her knowledge while automatizing the exploration towards the solution (i.e., instantiate the best ML pipeline). We now dig the foundation of HAMLET by incrementally introducing the concepts necessary to move from AutoML to Logic and Argumentation. To support the reader, we summarize the main notation in Table 1.

3.1. AutoML Formalization

We provide a novel formalization necessary to move from single algorithms to the optimal pipeline. For the sake of clarity, we refer to a Classification task, but the formalization also holds for supervised ML tasks in general.

Table 1: Main symbols used in the formalization.

Symbol	Meaning
A	Algorithm
h	Algorithm hyperparameter
S	Step
P	Pipeline
λ_*	Instance of *
Λ_*	Domain of *
Λ	Search space

Definition 1 (Dataset)**.** *A* dataset *X is a matrix where data items (i.e., rows) are characterized by features (i.e., columns).*

Definition 2 (Algorithm)**.** *An* algorithm *A is a function that transforms an input dataset X' into a new dataset X''. The algorithm exposes a (possibly empty) set H of* hyperparameters*. Each hyperparameter $h \in H$ has a* domain Λ_h*. We call the* algorithm domain Λ_A *the Cartesian product of all hyperparameter domains (i.e., $\Lambda_A = \Lambda_{h_1} \times \ldots \times \Lambda_{h_{|H|}}$). We call* algorithm instance $\lambda_A \in \Lambda_A$ *an algorithm whose hyperparameters have been assigned with values from their respective domains.*

A Classification algorithm returns a vector (i.e., a matrix with a single column) of labels Y out of the input dataset X'.

Definition 3 (Step)**.** *A* step *S is a set of alternative algorithms with the same goal. The* step domain *is defined as a disjoint union of the algorithm domains $\Lambda_S = \Lambda_{A_1} \cup \ldots \cup \Lambda_{A_{|S|}}$.*

Where $\cup$ combines the domains of the given algorithms, while retaining the original domain membership (i.e., it is possible to refer to the domain of each algorithm included in a step).

We identify two types of steps: Data Pre-preprocessing steps (e.g., Discretization, Normalization) shape the dataset for the last mandatory step, which fulfill the task—Classification in this case.

Example 1 (Algorithm and step)**.** *Examples of steps are Normalization ($\mathcal{N}$), Discretization ($\mathcal{D}$), and Classification (Cl). An algorithm for Classification is Decision Tree ($\mathcal{D}t$) [41], examples of hyperparameters for $\mathcal{D}t$ are its maximum* depth ($\mathbb{N}^+$) *and the minimum* samples split ($\mathbb{N}^+$) *required to split a node; hence $\Lambda_{\mathcal{D}t} = \mathbb{N}^+ \times \mathbb{N}^+$. An example of algorithm instance is $\lambda_{\mathcal{D}t} = \{\text{depth} = 3, \text{samples\_split} = 10\}$.*

Definition 4 (Pipeline)**.** *Given a (possibly empty) set of Pre-processing steps $\mathcal{S} = \{S_1, \ldots, S_{|\mathcal{S}|}\}$ and a Classification algorithm A from the Classification step, a* pipeline *P is a sequence that concatenates steps from $\mathcal{S}$ and A. The domain of a pipeline is $\Lambda_P = \Lambda_{S_1} \times \ldots \times \Lambda_{S_{|\mathcal{S}|}} \times \Lambda_A$. We call* pipeline instance *λ_P a sequence of algorithm instances $\lambda_P = \langle \lambda_{A_1}, \ldots, \lambda_{A_{|P|}} \rangle$ such that $\lambda_P \in \Lambda_P$.*

Example 2 (Pipeline and pipeline instance)**.** *Given the pre-processing steps Normalization ($\mathcal{N}$) and Discretization ($\mathcal{D}$), the possible pipelines for the DecisionTree ($\mathcal{D}t$) are:*

$$P_1 = \langle \mathcal{D}t \rangle \qquad P_2 = \langle \mathcal{D}, \mathcal{D}t \rangle \qquad P_4 = \langle \mathcal{D}, \mathcal{N}, \mathcal{D}t \rangle$$
$$P_3 = \langle \mathcal{N}, \mathcal{D}t \rangle \qquad P_5 = \langle \mathcal{N}, \mathcal{D}, \mathcal{D}t \rangle$$

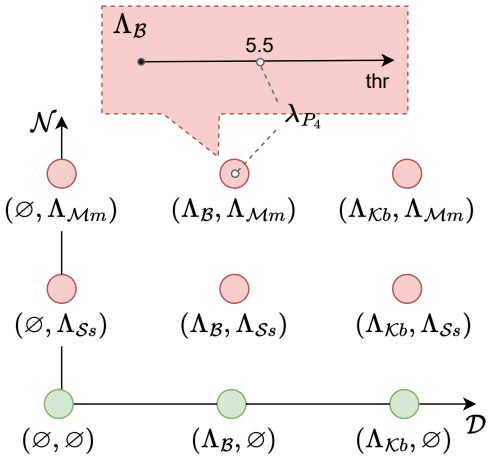


Figure 2: Examples of the pipeline domain Λ_{P_4} and pipeline instance λ_{P_4}, for the sake of visualization we omit the third dimension representing the domain of the Decision Tree. Green (or red) circles represent valid (or invalid) subregions of the search space; Normalization is not allowed in the pipeline. The rectangle represents a zoom in the domain of the Binarizer algorithm.

Given Binarizer *($\mathcal{B}$) and* KBinsDiscretizer *($\mathcal{K}b$) as algorithms of Discretization ($\mathcal{D}$), and* MinMaxScaler *($\mathcal{M}m$) and* StandardScaler *($\mathcal{S}s$) and as algorithms of Normalization ($\mathcal{N}$), we provide examples of instances of P_2 and P_4:*

$$\lambda_{P_2} = \langle \lambda_{\mathcal{B}},\ \lambda_{\mathcal{D}t} \rangle, \qquad \lambda_{P_4} = \langle \lambda_{\mathcal{K}b},\ \lambda_{\mathcal{M}m},\ \lambda_{\mathcal{D}t} \rangle$$
$$\lambda_{\mathcal{B}} = \{\text{thr} = 5.5\}, \qquad \lambda_{\mathcal{K}b} = \{\text{n\_bins} = 3, \ldots\}$$
$$\lambda_{\mathcal{M}m} = \{\varnothing\}, \qquad \lambda_{\mathcal{D}t} = \{\text{depth} = 3, \ldots\}$$

Figure 2 depicts the pipeline domain Λ_{P_4} and the pipeline instance λ_{P_4}.

Depending the on the involved algorithms, their order and hyperparameters, the search space – out of which the best pipeline instance is select – is defined as follows. While, its extraction is later discussed in Algorithm 1.

Definition 5 (Search space)**.** *The* search space Λ *is the Cartesian product of the domain of the Classification step and the disjoint union of the all partial permutations of the pre-preprocessing steps domains.*

AutoML optimizes the exploration of such space. However, it is not only about algorithms and hyperparameters but also about constraints.

Definition 6 (Constraint)**.** *A* constraint $C \subseteq \Lambda$ *is a region of search space that is either* mandatory *or* forbidden*. Given a pipeline instance $\lambda_P \in \Lambda_P \subseteq \Lambda$*

- *a mandatory constraint C is fulfilled if $\lambda_P \in C$;*
- *a forbidden constraint C is fulfilled if $\lambda_P \notin C$.*

Example 3 (Constraint)**.** *Given the Normalization step ($\mathcal{N}$) and Decision Tree ($\mathcal{D}t$) as a Classification algorithm, an example of algorithm-related constraint is "forbid $\mathcal{N}$ in pipelines with $\mathcal{D}t$". This discards all the pipelines containing both Normalization and Decision Tree. Figure 2 depicts the effects of the constraint on the pipeline domain Λ_{P_4}.*

Considering all the constraint combinations is overwhelming and, additionally, *conflicts* might occur; for instance in the case of ethical [42] and legal fields that easily inject conflicting constraints into the search space.

Definition 7 (Constrained pipelines optimization)**.** *Given a search space Λ and a set of constraints C, finding the best pipeline instance $\hat{\lambda}_P$ is defined as $\hat{\lambda}_P = argmax_{\lambda_P \in \Lambda_P} metric(\lambda_P)$, where $metric(\lambda_P)$ is the function evaluating the goodness of λ_P and the explored pipelines fulfill the constraints in C.*

3.2. Argumentation Formalization

AutoML is not explainable, hence it does not provide the DS with feedbacks that would help her to augment the knowledge about the problem. It is necessary to represent both (i) the DS knowledge about the problem and (ii) the outcome of the AutoML tool in a uniform human-readable medium. The former helps to drive the optimization process, the later augments the knowledge about the problem by learning from the explored configurations of pipeline instances—deriving new constraints that increase the DS awareness. We leverage Logic as the key element in defining a common structure (i.e., a uniformed human- and machine-readable medium) on which the knowledge of both the DS and the AutoML tool can be combined fruitfully. In a way, our approach follows the steps of the well known logical based expert systems, of which it is possible to find a great number of successful examples [43]. Logic provides the tools to cope with one of the distinctive features of the knowledge we want to deal with: inconsistency. Indeed, the ML process is the product of possible attempts, validated or refuted by a consequent evaluation. Hence, the mechanism used to encode the knowledge is required to manage this constant revision process. This is the role of Argumentation—one of the main approaches for dealing with inconsistent knowledge and defeasible reasoning.

Definition 8 (Argumentation Theory)**.** *An Argumentation theory is a tuple AS=$\langle L, R \rangle$ with:*

- *L an Argumentation language;*
- *R the set of defeasible rules in the form $r : \phi_0, \ldots, \phi_n \Rightarrow \phi$, where $\phi_0, \ldots, \phi_n, \phi$ are well-formed formulae in the L language and r is the identifier of the rule; we call $\phi_0, \ldots, \phi_n$ the premises of the rule, and ϕ its conclusion. Rules with no premises are allowed (i.e. $r :\Rightarrow \phi$).*

The set of rules R in the theory is used to define how elements from the language are combined together. In the following two definitions, we specialize L into the language L_{ML} expressing all the basic elements of an AutoML problem and R into a Logical Knowledge Base written in the language L_{ML}.

Definition 9 (AutoML language)**.** *Given an argumentation language L, we define the* AutoML language L_{ML} *as* $L \cup W$*, with W the following set of predicates*[2]*:*

- *step(S) with* $S \in L$*, representing a step S in the pipeline;*
- *algorithm(S, A) with* $S, A \in L$*, representing an algorithm A for the step S;*
- *hyperparameter(A, h, t) with* $A, h, t \in L$*, representing an hyperparameter h for the algorithm A of type t (e.g., numerical, categorical);*
- *domain(A, h,* Λ_h*) with* $A, h, \Lambda_h \in L$*, representing an hyperparameter h for the algorithm A with domain* Λ_h*;*
- *pipeline(*$\langle S_1, \ldots, S_n \rangle$*, A) with* $S_1, \ldots, S_n, A \in L$*, representing a pipeline consisting of the sequence of steps* $\langle S_1, \ldots, S_n \rangle$ *and the Classification algorithm A;*
- *mandatory(*$\langle S_1, \ldots, S_n \rangle$*, Z) with* $S_1, \ldots, S_n, Z \in L$*, representing a constraint imposing the steps* $\langle S_1, \ldots, S_n \rangle$ *on the pipelines with algorithm A (*$Z = A$*) or on all the Classification pipelines (*$Z = Cl$*);*
- *forbidden(*$\langle S_1, \ldots, S_n \rangle$*, Z) with* $S_1, \ldots, S_n, Z \in L$*, representing a constraint forbidding the steps* $\langle S_1, \ldots, S_n \rangle$ *on the pipelines with algorithm A (*$Z = A$*) or on all the Classification pipelines (*$Z = Cl$*);*
- *mandatory_order(*$\langle S_1, \ldots, S_n \rangle$*, Z) with* $S_1, \ldots, S_n, Z \in L$*, representing a constraint imposing the sequence of steps* $\langle S_1, \ldots, S_n \rangle$ *on the pipelines with algorithm A (*$Z = A$*) or on all the Classification pipelines (*$Z = Cl$*).*

Definition 10 (Logical Knowledge Base)**.** *Given the language* L_{ML}*, we call Logical Knowledge Base (LogicalKB) the set of rules for a given AutoML problem.*

In other words, the DS leverages an intuitive logical language (i.e., L_{ML}), and enlists the constraints one-by-one (i.e., in the LogicalKB). In our vision, the LogicalKB consists of (i) a set rules specified by the DS and a (ii) set of common rules that enable the automatic derivation of pipelines and constraints. Besides, the DS community could create a shared LogicalKB derived from the available literature and similar real-case problems.

Example 4 (Logical Knowledge Base)**.** *We focus on Discretization (*$\mathcal{D}$*), Normalization (*$\mathcal{N}$*) and Classification (Cl) steps, and, for brevity, only define the Classification algorithms: Decision Tree (*$\mathcal{D}t$*) and K-Nearest Neighbors (*$\mathcal{K}nn$*).*

```
# define Discretization step
s1 : ⇒ step(D).
# define Normalization step
s2 : ⇒ step(N).
# define Classification step
s3 : ⇒ step(Cl).
# DT is a Classification algorithm
a1 : ⇒ algorithm(Cl, Dt).
# Knn is a Classification algorithm
a2 : ⇒ algorithm(Cl, Knn).
# Forbid Normalization when using DT
c1 : ⇒ forbidden(⟨N⟩, Dt).
```

s1, s2, and s3 represent the steps; a1 and a2 represent the algorithms; finally, c1 represent the algorithm-related constraint from Example 3, namely "forbid $\mathcal{N}$ *in pipelines with* $\mathcal{D}t$*".*

When applying constraints, they can be conflicting. We reify the constraints from Definition 6 through conflict function in the Structured Argumentation domain-

Definition 11 (AutoML Conflict)**.** *The conflict function* c_{ML} *is a function from* L_{ML} *to* $2^{L_{ML}}$ *that given a statement from* L_{ML} *returns the set of conflicting statements.*

We support both the AutoML conflicts on "pipeline vs constraint" and "constraint vs constraint". Formally, let us consider two lists of steps $\alpha = \langle \ldots, S_i, S_j, \ldots \rangle$ and $\beta = \langle \ldots, S_y, S_x, \ldots \rangle$.

- Pipeline vs constraint: return the constraints conflicting with pipelines.

$$
\begin{aligned}
&c_{ML}(pipeline(\beta, A)) = \\
&\quad \{mandatory(\alpha, A) \mid \exists S_i \in \alpha \ s.t. \ S_i \notin \beta\} \cup \\
&\quad \{forbidden(\alpha, A) \mid \forall S_i \in \alpha, \ S_i \in \beta\} \cup \\
&\quad \{mandatory\_order(\alpha, A) \mid \exists S_i, S_j \in \alpha, S_x, S_y \in \beta, \\
&\qquad\qquad S_i = S_x, S_j = S_y \ s.t. \ i < j, x > y\}
\end{aligned}
$$

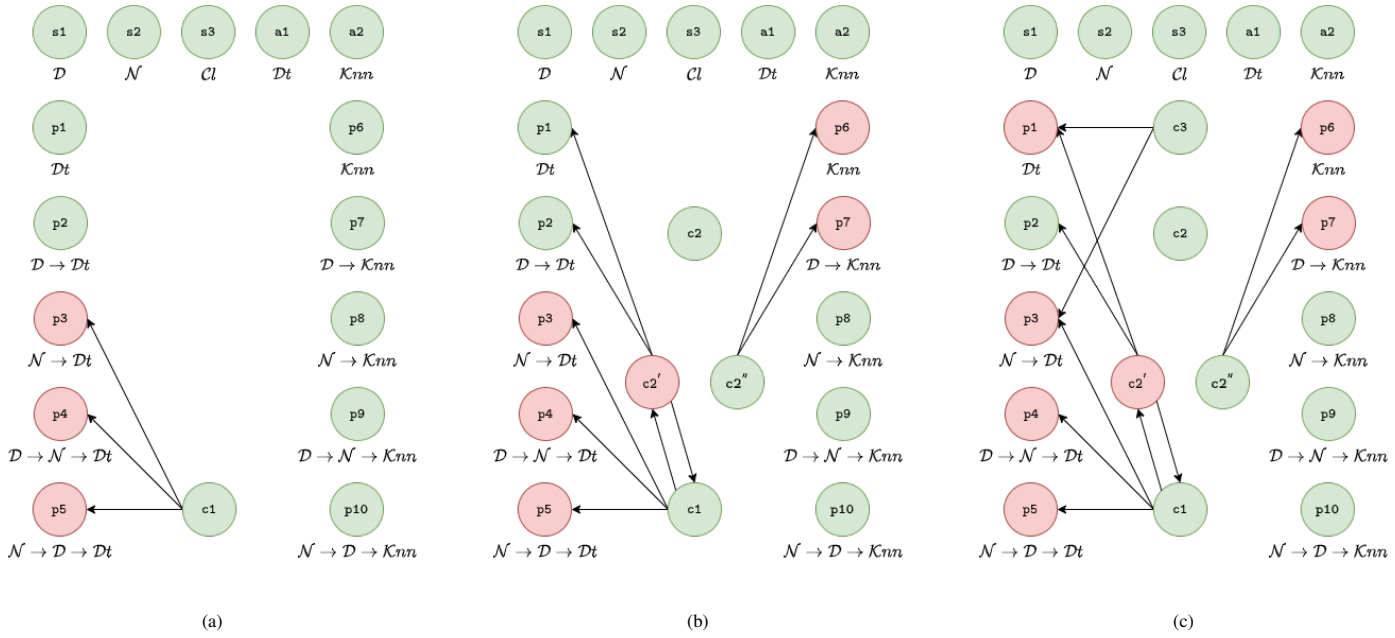
  Intuitively, a pipeline $pipeline(\langle S_i, S_j \rangle, A)$ is conflicting with a *mandatory* constraint if the pipeline does not contains at least a mandatory step (e.g., the pipeline is conflicting with $mandatory(\langle S_j, S_k \rangle, A)$), with a $forbidden$ constraint if the pipeline contains all the forbidden steps (e.g., the pipeline is conflicting with $forbidden(\langle S_j \rangle, A)$), and with a *mandatory_order* constraint if the pipeline contains at least two steps that are not in the mandatory order (e.g., the pipeline is conflicting with $mandatory\_order(\langle S_j, S_i \rangle, A)$).

- Constraint vs constraint: return the constraints conflicting with other constraints.

$$
\begin{aligned}
&c_{ML}(forbidden(\beta, A)) = \{mandatory(\alpha, A) \mid \forall S_j \in \beta, S_j \in \alpha\} \\
&c_{ML}(mandatory(\beta, A)) = \{forbidden(\alpha, A) \mid \forall S_j \in \alpha, S_j \in \beta\} \\
&c_{ML}(mandatory\_order(\beta, A)) = \\
&\quad \{mandatory\_order(\alpha, A) \mid \exists S_i, S_j \in \alpha, S_x, S_y \in \beta, \\
&\qquad\qquad S_i = S_x, S_j = S_y \ s.t. \ i < j, x > y\}
\end{aligned}
$$

  Intuitively, *mandatory* and $forbidden$ constraints are in conflict if all the forbidden steps are included in the mandatory constraint (i.e., $mandatory(\langle S_i, S_j, S_k \rangle, A)$) and a $forbidden(\langle S_i, S_j \rangle, A)$)), this hold symmetrically for $forbidden$ and *mandatory* constraints.

[2]For the sake of conciseness, when writing statements of the AutoML language, the letters S (and A) refer to the name of the step (and algorithm)



(a) (b) (c)

Figure 3: Examples of Problem Graphs. Green nodes are valid arguments, red ones are refuted. Arrows are attacks.

Two *mandatory_order* constraints are in conflict if they contain at least two steps in different order (i.e., *mandatory_order*$(\langle S_i, S_j, S_k \rangle, A)$) and a *mandatory_order*$(\langle S_j, S_i \rangle, A)$)).

Example 5 (AutoML conflict)**.** *With reference to the LogicalKB in Example 4, let us consider the set of rules that represent the pipelines related to* $\mathcal{D}t$*:*

```
# pipeline ending with a DT
p1 : ⇒ pipeline(Dt).
# Discretization and DT
p2 : ⇒ pipeline(⟨D⟩, Dt).
# Normalization and DT
p3 : ⇒ pipeline(⟨N⟩, Dt).
# Discretization, Normalization, and DT
p4 : ⇒ pipeline(⟨D, N⟩, Dt).
# Normalization, Discretization, and DT
p5 : ⇒ pipeline(⟨N, D⟩, Dt).
```

In this case, `c1` *(i.e., "forbid* $\mathcal{N}$ *in pipelines with* $\mathcal{D}t$*") is in conflict with the* pipeline *statements* `p3`*,* `p4`*, and* `p5` *since they contain* $\mathcal{N}$ *and* $\mathcal{D}t$*.*

To create a Problem Graph, which is the medium readable by users and machines, we informally (for the sake of conciseness) introduce the key elements from Structured Argumentation; a complete formalization is available in [39]. Given an Argumentation theory, an *argument* is created for every rule with no premises (e.g., from the rule $r :\Rightarrow c$, we derive an argument with *conclusion* c using r). Then, we recursively apply the other rules in the theory to the newly generated argument (e.g., we can use the argument with conclusion c and the following rule $r1 : c \Rightarrow d$ to conclude an argument for d using r and $r1$). This is repeated until no new argument can be generated. In the rest of the paper we will refer to an argument with the set of rules used to generate it (e.g., given $r :\Rightarrow c$ and $r1 : c \Rightarrow d$, we will write r and $r1$ when referring to the rules and $\{r\}$ and $\{r, r1\}$ for – respectively – the argument with *conclusion* c using r and the argument with *conclusion* d using r and $r1$).

An *Argumentation framework* is defined using the arguments built from an Argumentation theory and their *attack* relations. Attacks are inconsistencies between arguments' conclusions and are computed using a conflict function (Definition 11). For instance, given two arguments concluding respectively a and b, and the conflict function c such that $c(a) = \{b\}$, then the argument for b directly attacks the one for a (i.e., b is in conflict with a). The attack is propagated to all the arguments built over the receiver of the attack (e.g., if a_1 directly attacks a_2, and a_2's conclusion is used to derive a_3, then a_1 also attacks a_3). Also, we can define a *preference* relation over arguments using a partial ordering over the rules in the Argumentation theory: it is impossible for an argument to be attacked by the least preferred ones, even if they are in conflict. In other words, we can explicitly solve inconsistencies in the LogicalKB using priorities. We exploit the *last-weakest* ordering as in [39].

Finally, the evaluation of an Argumentation framework is performed through semantics, which determines all the sets of arguments that are consistent (called *extensions*) in an Argumentation framework. We exploit grounded semantics [36] to produce a *grounded extension*; this semantics is the most skeptical—i.e., it includes only the arguments that are verified by all the possible interpretations.

Definition 12 (Problem Graph)**.** *We call* Problem Graph *a graph in which nodes are arguments and edges are attacks from the Argumentation framework that is built on the Argumentation theory* $\langle L_{ML}, LogicalKB \rangle$ *and the conflict function* c_{ML}*.*

The benefits of the Problem Graph are two-fold. First of all, it can be leveraged by both DSs and domain experts to: under-

stand, summarize and visualize the current knowledge. Second of all, it is straightforward to convert such a graph of constraints into a space of possible solutions (i.e., exploiting Argumentation semantics, it is easy to obtain all the sets of arguments – constraints and pipelines – which hold together).

Example 6 (Problem Graph)**.** *Figure 3a illustrates the Problem Graph extracted from the LogicalKB introduced in Example 4 and 5 and evaluated under grounded semantics. Arguments are represented as nodes, attacks as arrows and the colors represent the state of the arguments according to the semantics: red for refuted arguments, and green for the ones in the extension. The arguments are identified through the set of rules used to build them. In the upper part of the figure, we have a group of undefeated arguments, namely* {s1}*,* {s2}*,* {s3}*,* {a1}*, and* {a2}*, representing the basic knowledge used to setup the AutoML search space (i.e. steps and algorithms). Then, we have an argument for every pipeline in Example 5: from* {p1} *to* {p5} *the pipelines regarding* $\mathcal{D}t$*, from* {p6} *to* {p10} *the ones regarding* $\mathcal{K}nn$*. Finally, we can observe three different attacks: from* {c1} *to* {p3}*,* {p4}*, and* {p5}*, in accordance with the conflicts identified in Example 5. The arguments in the extension give us all the information that we should use during the AutoML optimization process – i.e. we should discard all the pipelines refuted by the constraint argument (*{c1}*), and focus on the remaining part of the search space.*

The use of Argumentation relieves the DS of the burden of manually considering all the effects of the possible constraints. It is important to notice that, although the increased degree of automation, the Problem Graph allows the DS and domain experts to correct, revise, and supervise the process. Accordingly, possible inconsistencies – due to diverging constraints – can be verified by the DS using her knowledge.

Any change in the LogicalKB translates into a change in the Problem Graph, allowing the DS and domain experts to visualize it and argue about it. The revision of the Problem Graph is the key element in the process of augmenting the knowledge: the DS and domain experts can consult each other and discuss how the new insights relate to their initial knowledge. Indeed, thanks to the nature of the Problem Graph, it would be extremely easy to identify new possible conflicts and supporting arguments. Furthermore, AutoML can update the Problem Graph by extracting constraints from the performed exploration, and transposing them into the LogicalKB. For instance, the DS may not have considered that the dataset contains missing values. AutoML helps in identifying the new data-related constraint "require Imputation ($\mathcal{I}$) in all the pipelines" and adds it to the LogicalKB ($mandatory(\langle\mathcal{I}\rangle, Cl)$).

The described process is compliant with and augments the CRISP-DM process. The inferred/learned knowledge is automatically handled throughout iterations, supporting the DS in the whole analysis in a continuous revision of the constraints.

4. HAMLET

HAMLET iterates over three phases (Figure 1): (i) the generation of Problem Graph and search space out of the LogicalKB, (ii) the exploration of the search space in compliance with the specified constraints, and (iii) the augmentation of the LogicalKB through a rule recommendation.

The framework is available at `https://github.com/QueueInc/HAMLET`, and it is composed of two sub-modules. The first, written in Kotlin and running on the JVM, exposes a graphical interface on which the DSs can compile and revise the LogicalKB. The module is also responsible for the generation and evaluation of the Problem Graph; it implements the Structured Argumentation functionalities as specified in Section 3 using Arg2P [40], an ASPIC$^+$-based Kotlin library. The second module, written in Python, is responsible for performing the AutoML optimization and the extraction of the new constraints from the explored space.

4.1. Generation of Problem Graph and Search Space

In Section 3, we defined the LogicalKB as the set of rules specified by the DS using her knowledge. The LogicalKB also includes a set of hard-encoded rules representing inferences necessary to characterize the AutoML problems. These rules are joined to the ones defined by the DS and used to build the Problem Graph (i.e., Argumentation framework).

A subset of rules is shown in Figure 4. The first two ($hc0$ and $hc1$) define how to automatically derive a pipeline using algorithms and steps. The construction of pipelines can be completely automated and the DS should be dispensed from manually enumerating all the possible pipelines as in Example 5. In particular, the correct set of rules is built dynamically using the steps and algorithms provided by the DS, then they are used to derive all the arguments for the possible pipelines. The last three rules ($hc2, hc3$ and $hc4$) encode constraints – mandatory, forbidden, mandatory_order – on all the available algorithms with a single statement (e.g., $mandatory(\langle\mathcal{D}\rangle, Cl)$): it will be automatically used by the framework to derive the constraints for all the specific algorithms in the theory.

Example 7 (Hard-coded rules)**.** *With reference to Example 6 and Figure 4, we add rule* c2 *for a new data-related constraint.*

```
# mandatory Norm. in Class. pipelines
c2 : ⇒ mandatory(⟨N⟩, Cl)
```

From the rule c2*, the hard-coded rules generate the two arguments* c2' = {c2, a1, hc2} *(i.e.,* $mandatory(\langle\mathcal{N}\rangle, \mathcal{D}t)$*) and* c2'' = {c2, a2, hc2} *(i.e.,* $mandatory(\langle\mathcal{N}\rangle, \mathcal{K}nn)$*) that are specific for the Classification algorithms in the LogicalKB.*

However, {c1} *(i.e.,* $forbidden(\langle\mathcal{N}\rangle, \mathcal{D}t)$*; is in conflict with* c2'*. Depending on her experience, the DS decides to resolve the conflict by specifying an ordering over the rules in the LogicalKB. Assuming that the DS prefers* c1 *to* hc2*, the argument* {c1} *is preferred to* c2' *and the attack from the latter is not considered in the final graph. Figure 3b shows the updated graph. Firstly, we observe the support relation between* {c2} *and the*

```
# given an algorithm, create a pipeline including only such algorithm
hc0 : algorithm(Cl, A) ⇒ pipeline(⟨ ⟩, A).
# given some steps and an algorithm, create a pipeline including such steps and algorithm
hc1 : step(S_1),...,step(S_n), algorithm(Cl, A) ⇒ pipeline(⟨S_1,...,S_n⟩, A).
# given constraints on the Pre-processing steps required for Classification...
# ... apply this constraints to all Classification algorithms
hc2 : mandatory(⟨S_1,...,S_n⟩, Cl), algorithm(Cl, A) ⇒ mandatory(⟨S_1,...,S_n⟩, A).
hc3 : forbidden(⟨S_1,...,S_n⟩, Cl), algorithm(Cl, A) ⇒ forbidden(⟨S_1,...,S_n⟩, A).
hc4 : mandatory_order(⟨S_1,...,S_n⟩, Cl), algorithm(Cl, A) ⇒ mandatory_order(⟨S_1,...,S_n⟩, A).
```

Figure 4: A subset of rules from the LogicalKB.

generated constraints c2' and c2''. Since {c1} has no attackers, it is added to extension. Consequently, c2' is refuted and the the pipelines attacked by it are correctly reinstated.

Given the Problem Graph (we recall that the Problem Graph contains *all* the generated pipelines – including their partial permutations), the search space can be extracted as in Algorithm 1. We iterate over all the generated pipelines in the Problem Graph and we recursively build their domain: the pipeline domain is the Cartesian product of the step domains, the step domain is the disjoint union of the algorithm domains (we leverage the disjoint union since each algorithm can be picked as an alternative to the others), the algorithm domain is the Cartesian product of its hyperparameters; the domain of a hyperparameter is given by definition. Finally, the search space is the disjoint union of all the alternative pipeline domains.

Noticeably, while the search space could be constrained during its construction (e.g., by simply adding an "if" condition to check the validity of each pipeline at Algorithm 1 line 10), current AutoML frameworks leverage optimization techniques that do not allow the explicit exclusion of regions from the search space. As a consequence, we need to produce the entire search space first.

Algorithm 1 Search Space from the Problem Graph

Require: PG(N, E): Nodes and Edges of a Problem Graph
Ensure: Λ: Search Space

1: **procedure** GetDomain(A)
2: $\Lambda_A \leftarrow \varnothing$
3: **for each** $h \in A$ **do** ▷ For each hyperparameter in the algorithm...
4: $\Lambda_A \leftarrow \Lambda_A \times \Lambda_h$ ▷ Compute Cartesian product of hyperpar. domains
5: **end for**
6: **return** Λ_A ▷ Return the algorithm domain
7: **end procedure**

8: $\Lambda \leftarrow \varnothing$ ▷ Initialize the search space
9: **for each** $pipeline(\alpha, A) \in N$ **do** ▷ For each argument that is a pipeline with α steps and alg. A...
10: $\Lambda_P \leftarrow$ GetDomain(A) ▷ Init. pipeline domain with algorithm domain
11: **for each** $S \in \alpha$ **do** ▷ For each step in the pipeline...
12: $\Lambda_S \leftarrow \varnothing$ ▷ Init. the step domain
13: **for each** $A \in S$ **do** ▷ For each algorithm in the step...
14: $\Lambda_S \leftarrow \Lambda_S \cup$ GetDomain(A) ▷ Add alg. to step domain
15: **end for**
16: $\Lambda_P \leftarrow \Lambda_P \times \Lambda_S$ ▷ Add step domain to pipeline domain
17: **end for**
18: $\Lambda \leftarrow \Lambda \cup \Lambda_P$ ▷ Add pipeline domain to the search space
19: **end for**
20: **return** Λ ▷ Return the search space

4.2. Exploration of a Constrained Search Space

The Problem Graph is not only used to build the entire search space but it is also evaluated to understand which pipelines are invalid and which constraints are valid. Hence – through the Problem Graph – we enhance AutoML exploration by combining the following techniques.

(i) Invalid pipelines are used to discourage the exploration of such a portion of the search space (we recall that a pipeline has a domain – a region of the search space – in which several pipeline instances are parametrized). First, we sample such regions of the search space, then we enforce a knowledge injection mechanism through warm-starting (i.e., the process of providing previous evaluations that help the model to converge faster). For instance, with reference to Example 7, we sample some pipeline instances from the pipelines that have been discarded (from {p3} to {p7}); then, we label such samples as invalid and provide them to the AutoML tool, helping the optimization algorithm to focus only on the valid portions of the space.

(ii) Valid constraints – expressed as conjunctions of Boolean clauses – are used to discard the invalid pipeline instances that still are encountered by the AutoML tool. Indeed, since the sampling from (i) is non-exhaustive, it can happen that small portions of invalid regions could still be explored.

Our AutoML implementation is based on FLAML [44], which mixes Bayesian Optimization with CFO (Frugal Optimization for Cost-related Hyperparameters). In a standard Bayesian process, an increasingly accurate model is built on top of the previously explored pipeline instances to suggest the most promising ones among the remaining. The pipeline instances keep being explored, updating the model, until a budget in terms of either iterations or time is reached. With CFO, there is also an estimation of the evaluation time to consider the frugality of the suggested pipeline instances – hence favoring the ones requiring a smaller amount of time. Throughout the exploration, different solutions are tested, which contribute to augmenting the global knowledge about the problem.

4.3. Knowledge Augmentation through Rule Recommendation

New constraints are automatically mined out of the pipeline instances explored by AutoML and *recommended* in our logical language as rules. Then, the DS decides which rules are accepted and added to the LogicalKB.

At this stage, we leverage frequent pattern mining techniques to learn constraints in an unsupervised manner. Frequent pattern mining is the task of finding the most frequent and relevant patterns in large datasets (e.g., finding the products frequently bought together in the domain of market basket analysis); depending on the constraint type, we look for (sub)sets [45] or (sub)sequences [46] frequently recurring among the explored pipelines. Since a pipeline instance is a sequence of algorithms, the set of the explored pipeline instances can be directly mapped into a transactional dataset [45] where each pipeline instance is a transaction and each step – inferred from the algorithm – is an item.

We recommend the same constraints we support at the Argumentation level (i.e., *mandatory*, $forbidden$, *mandatory_order*) so that AutoML can be as expressive as the DS. For mandatory and forbidden constraints we look for (sub)sets [45] frequently recurring among the explored pipelines. Specifically, we split the explored pipeline instances by the applied Classification algorithm, set a minimum frequency (i.e., support) threshold to 50% (i.e., to be retrieved, a set/sequence must occur at least in 50% of the explored instances), and extract frequent maximal[3] itemsets. The recommendation depends on the constraint.

- *mandatory*: we consider only the patterns with good performance (i.e., $0.7 \leq metric \leq 1.0$);
- $forbidden$: we consider only the patterns with bad performance (i.e., $0.0 \leq metric \leq 0.3$);
- *mandatory_order*: the same considerations of the mandatory constraints stand, except that we look for (sub)sequences [46] of length 2 to discover ordering dependencies in pairs of steps as in [47].

We leveraged well-known implementations [48] and [49] for itemsets and sequences mining, respectively. Finally, we return to the DS only the top-10 rules sorted by descending support; we allow the DS to explore all the rules on-demand.

The thresholds act as filters on the extracted rules since we cannot burden the user with the investigation of hundreds of recommendations. As to the intervals, our rationale is simple: we only want to recommend as mandatory (order) the rules that achieved "good performance" and as forbidden the rules that achieved "bad performance". Since we handle classification pipelines that mainly refer to (balanced) accuracy/F1 score/recall, we mapped "good" in the interval $[0.7, 1.0]$ and "bad" in the interval $[0, 0.3]$. For the frequent pattern extraction, we consider only the pipeline instances falling in these intervals. As to the support, 50% ensures that the pattern recurs on many of the explored instances and empirically showed to be a good threshold to have good efficiency in the extraction of frequent patterns.

Example 8 (Rules Recommendation)**.** *With reference to the Problem Graph in Example 7, the AutoML results are filtered according to the chosen metric, the algorithm [48] is applied, and let us assume that the rule* `c3` *is recommended:*

```
c3 : ⇒ mandatory(⟨D⟩, Dt).
```

The constraints specifies "mandatory $\mathcal{D}$ in pipelines with $\mathcal{D}t$". As a matter of fact, it is well known that Discretization improves the performance of tree-based algorithms giving to them the ability to apply multiple split in the decision nodes. Figure 3c shows the effect of the applied constraint: a new portion of the search space is excluded from the extension ($\{p1\}$).

5. Experimental Evaluation

The performance of HAMLET depends on (i) the rules encoded in the LogicalKB and (ii) the rules recommended after each run. To test both the effectiveness and efficiency of our approach, we define three experimental settings.

- PKB (Preliminary Knowledge Base), HAMLET starts with a preliminary LogicalKB constraining the search space from the first iteration, and no rule mining is applied. The preliminary LogicalKB consists of the rules discovered in [47] and some well-known from the literature (e.g., suggested by scikit-learn[4]). The complete knowledge base can be found in the Github repository.
- IKA (Iterative Knowledge Augmentation), HAMLET starts with an empty LogicalKB, and all the rules recommended after each run are applied to extend the LogicalKB.
- PKB+IKA, HAMLET starts with a preliminary LogicalKB, and the rules recommended after each run are applied to extend the LogicalKB.

HAMLET run 4 times in every setting – intuitively, four runs of knowledge augmentation – the budget assigned to each run is 125 pipeline instances in 900 seconds (15 minutes). We also test against a baseline: we let AutoML explore 500 pipeline instances ($= 125 \cdot 4$) in a single run with a time budget of 3600 seconds ($= 900 \cdot 4$; 1 hour).

For such an evaluation, we derive a search space out of 6 steps, 5 Data Pre-processing steps (Imputation, Normalization, Discretization, Feature Engineering, and Rebalancing) followed by the final Classification task. Since the tests are run on datasets from OpenML [50] – a well-known repository for data acquisition and benchmarking – and it provides already-encoded datasets, we do not consider the encoding step. Except

[3] Maximal itemsets are patterns that are not contained in any other. For instance, given two frequent patterns, $\{a, b, c\}$ and $\{a, b\}$, the former is maximal while the latter is not.

[4] `https://scikit-learn.org/stable/auto_examples/preprocessing/plot_discretization.html`

Table 2: Algorithms and number of hyperparameters for each of the steps in HAMLET. Algorithm names and hyperparameters are imported from the scikit-learn Python library.

Step	Algorithm	#Hyperparameters
Imputation	SimpleImputer	1
	IterativeImputer	2
Normalization	StandardScaler	2
	MinMaxScaler	0
	RobustScaler	2
	PowerTransformer	0
Discretization	Binarizer	1
	KBinsDiscretizer	3
Feature Eng.	SelectKBest	1
	PCA	1
Rebalancing	NearMiss	1
	SMOTE	1
Classification	DecisionTreeClassifier	7
	KNeighborsClassifier	3
	RandomForestClassifier	7
	AdaBoostClassifier	2
	MLPClassifier	6

Table 3: Dataset descriptions.

OpenMLID [a]	Dataset	Instances		Features		Classes	
40983	wilt	4839	L	6	L	2	L
40499	texture	5500	L	41	L	11	H
1485	madelon	2600	L	501	H	2	L
1478	har	10229	L	562	H	6	H
1590	adult	48842	H	9	L	2	L
–	–	–	H	–	L	–	H
–	–	–	H	–	H	–	L
554	mnist_784	70000	H	785	H	10	H

– Not Applicable
H The value v is high for the meta-feature F if $v \geq \frac{1}{|F|}\sum_{f \in F} f$
L The value v is low for the meta-feature F if $v < \frac{1}{|F|}\sum_{f \in F} f$
[a] Datasets are available at `https://www.openml.org/d/<OpenMLID>`

for that, we included all the Data Pre-processing steps and algorithms available in the scikit-learn [51] Python library (plus imbalance-learn [52] for Rebalancing transformations). The leveraged steps, algorithms per step, and hyperparameters per algorithm are reported in Table 2.

The OpenML-CC18 suite is a well-known collection of 72 datasets for benchmarking. Given the time-consuming computation of each dataset (8 hours per dataset = 2 hours for the baseline + 6 hours for HAMLET in the three settings) – in this preliminary evaluation – we select a representative subset of datasets according to three meta-features provided by OpenML: number of instances, number of features, and number of classes. For each of the considered meta-features, we search for datasets with either high or low values, and we select the representatives that maximize the overall dataset diversification. Table 3 illustrates the 6 datasets that have been identified; note that some combinations of meta-features have no representative dataset in the suite. Among these, we do not report the results for the dataset mnist_784 since the number of explored pipeline instances is insufficient to validate the result (i.e., due to the time necessary to run a single pipeline instance, only 50 instances were explored out of 1000).

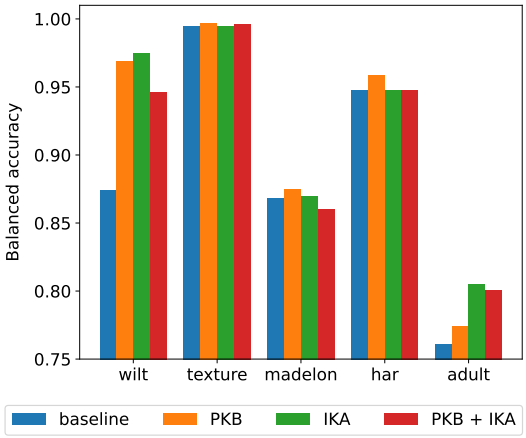


Figure 5: Results assessing the effectiveness of HAMLET w.r.t. the baseline.

5.1. *Effectiveness*

We employ balanced accuracy as the quality metric. For instance, in case of (two) binary classes, such a score is

$$\texttt{Balanced accuracy} = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right)$$

where TP and TN stand respectively for True Positive and True Negative (i.e., number of instances that have been correctly assigned to the positive and negative classes), and FP and FN stand respectively for False Positive and False Negative (i.e., number of instances that have been mistakenly assigned to the positive and negative classes). The formulation generalized to more than 2 classes can be found at [53]. The score avoids inflated performance estimations on imbalanced datasets. For balanced datasets, the score is equal to the conventional accuracy (i.e., the number of correct predictions divided by the total number of predictions), otherwise it drops to $\frac{1}{\#classes}$.

Figure 5 illustrates the performance achieved by the baseline and the three settings of HAMLET. HAMLET is clearly beneficial since in all datasets the framework overcomes the baseline. The preliminary results highlight that both the LogicalKB and rule recommendation play important roles:

- When we warm-start the exploration with a non-empty LogicalKB (PKB), in all datasets HAMLET overcomes the baseline.
- When we only leverage rule recommendation (IKA), we achieve results that are better than or equivalent to PKB, indeed we are injecting in the LogicalKB new rules that are tailored to the dataset.
- The synergy of PKB+IKA performs better than PKB in adult, worse in wilt, and the two are comparable in the other datasets. On the one hand, the PKB act as a warm start mechanism that speeds up the optimization; on the other hand, if not aligned with the recommended rules, it can mitigate the benefits of IKA. This proves to be a promising direction that further requires investigation since merging the words will require further studies. Indeed, it is worth noting that the recommended rules can

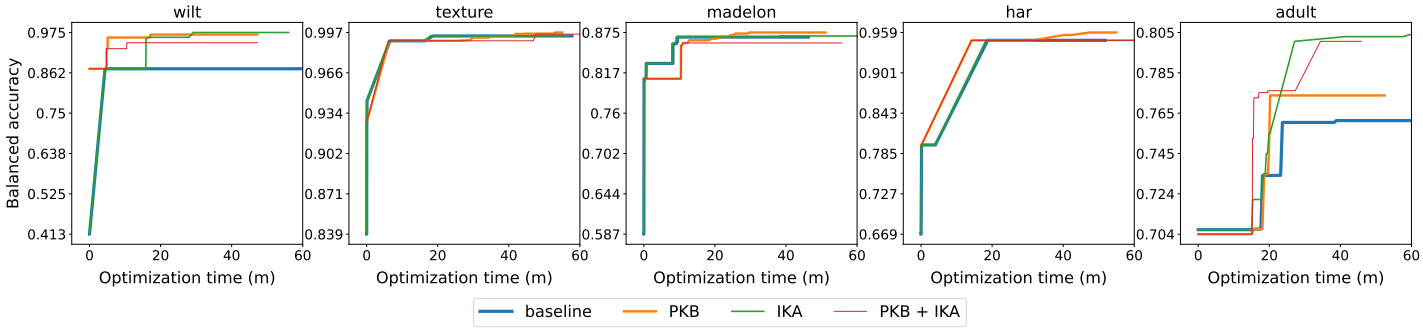


Figure 6: Results assessing the performance of HAMLET through the optimization time.

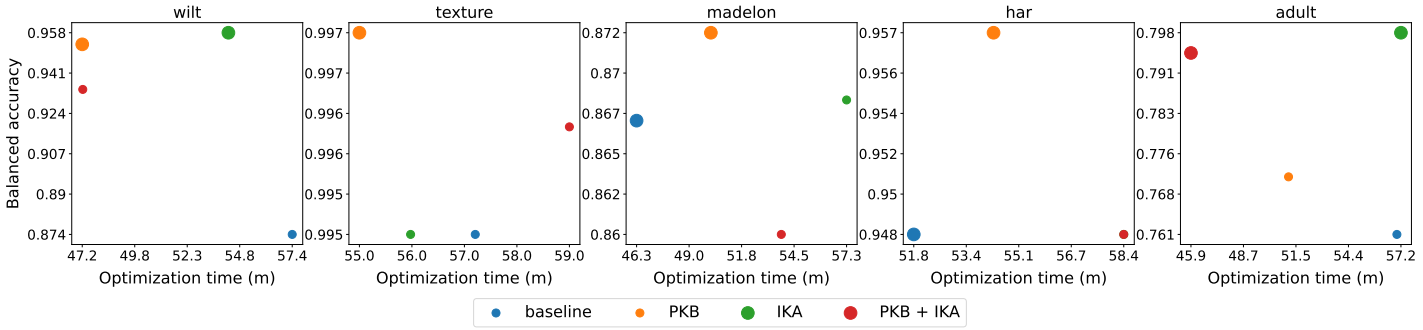


Figure 7: Comparison of the best pipeline instances characterized by optimization time and (balanced) accuracy, bigger circles represent settings that dominate the others.

be overlapping with the ones in the LogicalKB, highlighting the need to improve the recommendation process by also considering the rules that are already present in the LogicalKB.

In PKB+IKA, IKA can introduce rules that contradict the ones in the LogicalKB of PKB; for instance when the PKB contains rules that are not "representative" of the dataset/algorithms in use. We believe that this is an added value of HAMLET since "incomplete" (or even wrong) LogicalKBs can be corrected/refined by a data-driven approach. Finally, PKB+IKA and IKA are likely to produce different rules, since in PKB+IKA the LogicalKB biases the exploration of the search space from the beginning (acting as a warm start mechanism).

5.2. *Efficiency*

Figure 6 shows how settings converge to the optimal pipeline instance. Noticeably, PKB and PKB+IKA start with higher accuracy than IKA and the baseline in four datasets out of five, proving how the preliminary LogicalKB warm starts the exploration. However, time and #iterations alone are not fair metrics for comparison; for instance, an optimization strategy could privilege simple algorithms taking small amounts of computational time but producing worse results than "more complex" algorithms. In the direction of multi-objective optimization (exploration time should be minimized while accuracy should be maximized), Figure 7 depicts which settings dominate the others using the Skyline operator [54]. A setting dominates another one if it is as good or better in all dimensions (time and accuracy) and better in at least one dimension (time or accuracy). PKB dominates in 80% of the datasets, IKA in 40%, PKB+IKA in 20%, and the baseline in 40%. Noticeably, the baseline is selected as dominating only in madelon and har datasets due to the fact that converges faster than HAMLET (although it converges to a pipeline instance with lower accuracy).

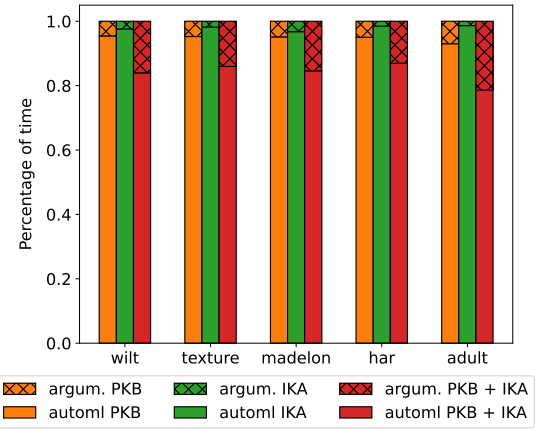


Figure 8: Computational time of the argumentation and AutoML processes.

Finally, Figure 8 depicts the overhead introduced by the argumentation framework in HAMLET that, at maximum, is 20% of the computational time in the adult dataset. This proves that the argumentation time is marginal with respect to the duration of the optimization process. As expected, PKB+IKA shows the highest overhead since the number of rules to manage is the highest.

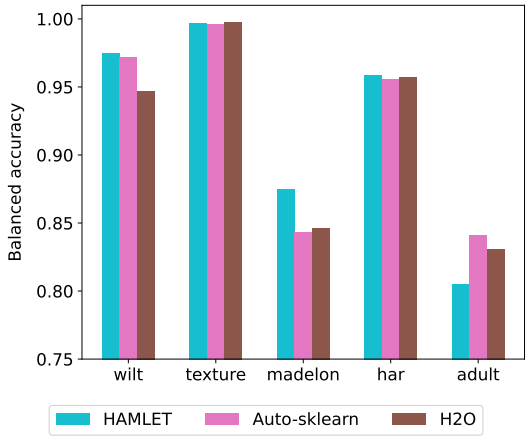


Figure 9: Results assessing the performance of HAMLET w.r.t. Auto-sklearn [19] and H2O [55].

5.3. Comparison

Figure 9 compares HAMLET against two well-known AutoML frameworks: Auto-sklearn [19] and H2O [55]. In four datasets out of five, HAMLET outperforms or is comparable to the two frameworks. Additionally, the added value of HAMLET is *explainability*. Hamlet is a human-in-the-loop AutoML framework tailored to the needs of DS that (i) enables the injection of their experience into the exploration process as well as (ii) the spreading and sharing of knowledge bases that encode what DSs have understood by the optimization of their pipelines.

6. Conclusions and Future Work

Data platforms support Data Scientists in performing end-to-end data analysis; to this end, Machine Learning plays a primary role. However, the complexity and heterogeneity of (Automated) Machine Learning processes are leading Data Scientists to lose control over such processes. Human awareness about the constraints and solutions of Machine Learning tasks is a fundamental aspect to consider, and consequently, the Data Scientist should play a central role in the design of next-generation data platforms.

According to this vision, we present HAMLET, a framework for Human-centered AutoML based on Logic and Structured Argumentation. Logic is exploited to structure the knowledge that the Data Scientist gathers while designing, modeling, and deploying a solution. The logical encoding of the knowledge provides a medium that is both human- and machine-readable and it allows an easy exploration and verification of all the constraints that may apply to the case at hand—it is overwhelming for the Data Scientist to correctly handle the vast amount of them. The preliminary evaluation of HAMLET shows promising results against state-of-the-art AutoML algorithms both in terms of effectiveness and efficiency, with argumentation introducing a small overhead with respect to the duration of the exploration process.

The directions for future work are plentiful, among them:

(i) the recommendation of more constraints out of the explored search space (e.g., rare or negative patterns);

(ii) the support to heterogeneous constraints on hyperparameter domains;

(iii) the injection of meta-learning into our Logical Knowledge Base to better identify when and how the constraints should be applied (e.g., this can be done after testing HAMLET on a multitude of datasets);

(iv) the introduction of a visual metaphor (e.g., based on the Problem Graph) to help Data Scientists' understanding;

(v) the study of automatic resolution/recommendation of conflicting constraints, also depending on the rules already embedded in the Logical Knowledge Base.

References

[1] M. Francia, E. Gallinucci, M. Golfarelli, A. G. Leoni, S. Rizzi, N. Santolini, Making data platforms smarter with MOSES, Future Gener. Comput. Syst. 125 (2021) 299–313. doi:10.1016/j.future.2021.06.031.
URL https://doi.org/10.1016/j.future.2021.06.031

[2] C. Forresi, M. Francia, E. Gallinucci, M. Golfarelli, Optimizing execution plans in a multistore, in: L. Bellatreche, M. Dumas, P. Karras, R. Matulevičius (Eds.), Advances in Databases and Information Systems, Springer International Publishing, Cham, 2021, pp. 136–151.

[3] P. Agrawal, R. Arya, A. Bindal, S. Bhatia, A. Gagneja, J. Godlewski, Y. Low, T. Muss, M. M. Paliwal, S. Raman, V. Shah, B. Shen, L. Sugden, K. Zhao, M.-C. Wu, Data platform for machine learning, in: Proceedings of the 2019 International Conference on Management of Data, SIGMOD '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 1803–1816. doi:10.1145/3299869.3314050.
URL https://doi.org/10.1145/3299869.3314050

[4] L. Zhou, S. Pan, J. Wang, A. V. Vasilakos, Machine learning on big data: Opportunities and challenges, Neurocomputing 237 (2017) 350–361. doi:10.1016/j.neucom.2017.01.026.

[5] R. Wirth, J. Hipp, Crisp-dm: Towards a standard process model for data mining, in: Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining, Vol. 1, Springer-Verlag London, UK, 2000.

[6] D. Xin, E. Y. Wu, D. J. L. Lee, N. Salehi, A. G. Parameswaran, Whither automl? understanding the role of automation in machine learning workflows, in: CHI '21: CHI Conference on Human Factors in Computing Systems, ACM, 2021, pp. 83:1–83:16. doi:10.1145/3411764.3445306.

[7] Y. Gil, J. Honaker, S. Gupta, Y. Ma, V. D'Orazio, D. Garijo, S. Gadewar, Q. Yang, N. Jahanshad, Towards human-guided machine learning, in: Proceedings of the 24th International Conference on Intelligent User Interfaces, 2019, pp. 614–624.

[8] D. J.-L. Lee, S. Macke, A human-in-the-loop perspective on automl: Milestones and the road ahead, IEEE Data Engineering Bulletin (2020).

[9] D. Wang, J. D. Weisz, M. Muller, P. Ram, W. Geyer, C. Dugan, Y. Tausczik, H. Samulowitz, A. Gray, Human-ai collaboration in data science: Exploring data scientists' perceptions of automated ai, Proceedings of the ACM on Human-Computer Interaction 3 (CSCW) (2019) 1–24.

[10] M. Francia, M. Golfarelli, S. Rizzi, Augmented business intelligence, in: I. Song, O. Romero, R. Wrembel (Eds.), Proceedings of the 21st International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data, co-located with EDBT/ICDT Joint Conference, DOLAP@EDBT/ICDT 2019, Lisbon, Portugal, March 26, 2019, Vol. 2324 of CEUR Workshop Proceedings, CEUR-WS.org, 2019.
URL http://ceur-ws.org/Vol-2324/Paper02-MGolfarelli.pdf

[11] M. Francia, E. Gallinucci, M. Golfarelli, COOL: A framework for conversational OLAP, Inf. Syst. 104 (2022) 101752. doi:10.1016/j.is.2021.101752.
URL https://doi.org/10.1016/j.is.2021.101752

[12] J. Drozdal, J. Weisz, D. Wang, G. Dass, B. Yao, C. Zhao, M. Muller, L. Ju, H. Su, Trust in automl: exploring information needs for establishing trust in automated machine learning systems, in: Proceedings of the 25th International Conference on Intelligent User Interfaces, 2020, pp. 297–307.

[13] J. P. Ono, S. Castelo, R. Lopez, E. Bertini, J. Freire, C. T. Silva, Pipelineprofiler: A visual analytics tool for the exploration of automl pipelines, IEEE Transactions on Visualization and Computer Graphics 27 (2) (2021) 390–400.

[14] J. Giovanelli, G. Pisano, Towards human-centric automl via logic and argumentation, in: M. Ramanath, T. Palpanas (Eds.), Proceedings of the Workshops of the EDBT/ICDT 2022 Joint Conference, Edinburgh, UK, March 29, 2022, Vol. 3135 of CEUR Workshop Proceedings, CEUR-WS.org, 2022.
URL http://ceur-ws.org/Vol-3135/dataplat_short2.pdf

[15] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, K. Leyton-Brown, Auto-weka: Automatic model selection and hyperparameter optimization in weka, in: Automated Machine Learning, Springer, Cham, 2019, pp. 81–95.

[16] P. I. Frazier, A tutorial on bayesian optimization, CoRR abs/1807.02811 (2018). arXiv:1807.02811.
URL http://arxiv.org/abs/1807.02811

[17] J. Giovanelli, B. Bilalli, A. Abelló, Effective data pre-processing for automl, in: Proceedings of the 23rd International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data (DOLAP), Vol. 2840 of CEUR Workshop Proceedings, CEUR-WS.org, 2021, pp. 1–10.

[18] A. Quemy, Data pipeline selection and optimization., in: DOLAP, 2019.

[19] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, F. Hutter, Auto-sklearn: efficient and robust automated machine learning, in: Automated Machine Learning, Springer, Cham, 2019, pp. 113–134.

[20] S. Falkner, A. Klein, F. Hutter, Bohb: Robust and efficient hyperparameter optimization at scale, in: International Conference on Machine Learning, PMLR, 2018, pp. 1437–1446.

[21] Q. Wang, Y. Ming, Z. Jin, Q. Shen, D. Liu, M. J. Smith, K. Veeramachaneni, H. Qu, Atmseer: Increasing transparency and controllability in automated machine learning, in: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, 2019, pp. 1–12.

[22] V. Perrone, M. Donini, M. B. Zafar, R. Schmucker, K. Kenthapadi, C. Archambeau, Fair bayesian optimization, in: Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society, 2021, pp. 854–863.

[23] M. Yaghini, A. Krause, H. Heidari, A human-in-the-loop framework to construct context-aware mathematical notions of outcome fairness, in: Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society, 2021, pp. 1023–1033.

[24] D. Peng, X. Dong, E. Real, M. Tan, Y. Lu, G. Bender, H. Liu, A. Kraft, C. Liang, Q. Le, Pyglove: Symbolic programming for automated machine learning, Advances in Neural Information Processing Systems 33 (2020) 96–108.

[25] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, M. I. Jordan, Mlbase: A distributed machine-learning system., in: Cidr, Vol. 1, 2013, pp. 2–1.

[26] M. Vartak, P. Ortiz, K. Siegel, H. Subramanyam, S. Madden, M. Zaharia, Supporting fast iteration in model building, in: NIPS Workshop LearningSys, 2015, pp. 1–6.

[27] D. Xin, L. Ma, J. Liu, S. Macke, S. Song, A. Parameswaran, Accelerating human-in-the-loop machine learning: Challenges and opportunities, in: Proceedings of the second workshop on data management for end-to-end machine learning, 2018, pp. 1–4.

[28] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, D. Sculley, Google vizier: A service for black-box optimization, in: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, 2017, pp. 1487–1495.

[29] D. Wang, P. Ram, D. K. I. Weidele, S. Liu, M. Muller, J. D. Weisz, A. Valente, A. Chaudhary, D. Torres, H. Samulowitz, et al., Autoai: Automating the end-to-end ai lifecycle with humans-in-the-loop, in: Proceedings of the 25th International Conference on Intelligent User Interfaces Companion, 2020, pp. 77–78.

[30] D. Wang, J. Andres, J. D. Weisz, E. Oduor, C. Dugan, Autods: Towards human-centered automation of data science, in: Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, 2021, pp. 1–12.

[31] M. Muller, I. Lange, D. Wang, D. Piorkowski, J. Tsay, Q. V. Liao, C. Dugan, T. Erickson, How data science workers work with data: Discovery, capture, curation, design, creation, in: Proceedings of the 2019 CHI conference on human factors in computing systems, 2019, pp. 1–15.

[32] F. Pfisterer, J. Thomas, B. Bischl, Towards human centered automl, arXiv preprint arXiv:1911.02391 (2019).

[33] T. T. Khuat, D. J. Kedziora, B. Gabrys, The roles and modes of human interactions with automated machine learning systems, arXiv preprint arXiv:2205.04139 (2022).

[34] A. Crisan, B. Fiore-Gartland, Fits and starts: Enterprise use of automl and the role of humans in the loop, in: Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, 2021, pp. 1–15.

[35] L. C. Paulson, Computational logic: its origins and applications, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences 474 (2210) (Feb 2018). doi:10.1098/rspa.2017.0872.

[36] P. M. Dung, On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games, Artificial Intelligence 77 (2) (1995) 321–358. doi:10.1016/0004-3702(94)00041-X.

[37] P. Baroni, M. Caminada, M. Giacomin, An introduction to argumentation semantics, Knowledge Engineering Review 26 (4) (2011) 365–410. doi:10.1017/S0269888911000166.

[38] P. Besnard, A. J. García, A. Hunter, S. Modgil, H. Prakken, G. R. Simari, F. Toni, Introduction to structured argumentation, Argument & Computation 5 (1) (2014) 1–4. doi:10.1080/19462166.2013.869764.

[39] S. Modgil, H. Prakken, The *ASPIC*$^+$ framework for structured argumentation: a tutorial, Argument & Computation 5 (1) (2014) 31–62. doi:10.1080/19462166.2013.869766.

[40] R. Calegari, G. Pisano, A. Omicini, G. Sartor, Arg2P: An argumentation framework for explainable intelligent systems, Journal of Logic and Computation (December 2021). doi:10.1093/logcom/exab089.

[41] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone, Classification and Regression Trees, Wadsworth, 1984.

[42] D. Harrison, D. Rubinfeld, Hedonic housing prices and the demand for clean air, Journal of Environmental Economics and Management 5 (1978) 81–102. doi:10.1016/0095-0696(78)90006-2.

[43] H. Tan, A brief history and technical review of the expert system research, IOP Conference Series: Materials Science and Engineering 242 (9 2017). doi:10.1088/1757-899X/242/1/012111.

[44] C. Wang, Q. Wu, M. Weimer, E. Zhu, Flaml: A fast and lightweight automl library, Proceedings of Machine Learning and Systems 3 (2021) 434–447.

[45] R. Srikant, R. Agrawal, Mining generalized association rules (1995).

[46] R. Srikant, R. Agrawal, Mining sequential patterns: Generalizations and performance improvements, in: International conference on extending database technology, Springer, 1996, pp. 1–17.

[47] J. Giovanelli, B. Bilalli, A. Abelló, Data pre-processing pipeline generation for autoetl, Information Systems (2021) 101957.

[48] S. Raschka, Mlxtend: Providing machine learning and data science utilities and extensions to python's scientific computing stack, The Journal of Open Source Software 3 (24) (Apr. 2018). doi:10.21105/joss.00638.
URL http://joss.theoj.org/papers/10.21105/joss.00638

[49] X. Wang, A. Hosseininasab, P. Colunga, S. Kadioglu, W.-J. van Hoeve, Seq2pat: Sequence-to-pattern generation for constraint-based sequential pattern mining, Proceedings of the AAAI Conference on Artificial Intelligence TBD (TBD) (2022) TBD.
URL https://github.com/fidelity/textwiser

[50] J. Vanschoren, J. N. van Rijn, B. Bischl, L. Torgo, Openml: networked science in machine learning, SIGKDD Explorations 15 (2) (2013) 49–60. doi:10.1145/2641190.2641198.
URL http://doi.acm.org/10.1145/2641190.264119

[51] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[52] G. Lemaître, F. Nogueira, C. K. Aridas, Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning, Journal of Machine Learning Research 18 (17) (2017) 1–5.
URL http://jmlr.org/papers/v18/16-365.html

[53] K. H. Brodersen, C. S. Ong, K. E. Stephan, J. M. Buhmann, The balanced accuracy and its posterior distribution, in: 20th International Conference on Pattern Recognition, ICPR 2010, Istanbul, Turkey, 23-26 August 2010, IEEE Computer Society, 2010, pp. 3121–3124. doi:10.1109/ICPR.2010.764.
URL https://doi.org/10.1109/ICPR.2010.764

[54] S. Borzsony, D. Kossmann, K. Stocker, The skyline operator, in: Proceedings 17th international conference on data engineering, IEEE, 2001, pp. 421–430.

[55] E. LeDell, S. Poirier, H2o automl: Scalable automatic machine learning, in: Proceedings of the AutoML Workshop at ICML, Vol. 2020, 2020.