On the Design of PSyKE: A Platform for Symbolic Knowledge Extraction

Federico Sabbatini[1], Giovanni Ciatto[1], Roberta Calegari[2] and Andrea Omicini[1]

[1]*Dipartimento di Informatica – Scienza e Ingegneria (DISI), Alma Mater Studiorum—Università di Bologna, Italy*
[2]*Alma Mater Research Institute for Human-Centered Artificial Intelligence, Alma Mater Studiorum—Università di Bologna, Italy*

Abstract

A common practice in modern explainable AI is to *post-hoc* explain black-box machine learning (ML) predictors – such as neural networks – by extracting *symbolic* knowledge out of them, in the form of either rule lists or decision trees. By acting as a surrogate model, the extracted knowledge aims at revealing the inner working of the black box, thus enabling its inspection, representation, and explanation.

Various knowledge-extraction algorithms have been presented in the literature so far. Unfortunately, running implementations of most of them are currently either proof of concepts or unavailable. In any case, a unified, coherent software framework supporting them all – as well as their interchange, comparison, and exploitation in arbitrary ML workflows – is currently missing.

Accordingly, in this paper we present PSyKE, a platform providing general-purpose support to symbolic knowledge extraction from different sorts of black-box predictors via many extraction algorithms. Notably, PSyKE targets symbolic knowledge in *logic form*, allowing the extraction of first-order logic clauses. The extracted knowledge is thus both machine- and human-interpretable, and can be used as a starting point for further symbolic processing—e.g. automated reasoning.

Keywords

explainable AI, knowledge extraction, interpretable prediction, PSyKE

1. Introduction

Artificial neural networks (ANN), support vector machines (SVM), and other data-driven predictors are nowadays among the most-used tools to face a wide range of different tasks involving machines learning (ML) from data [1]. In all those cases, the learning activity consists of tuning the parameters of predefined algorithms in order to maximise their predictive capability w.r.t. the data at hand.

The major drawback of state-of-the-art ML algorithms is that they are inherently *opaque*, meaning that they do not provide any intelligible representation of what they learn from data. This is why most of those algorithms are considered as black boxes (BB) which only represent

WOA 2021: Workshop "From Objects to Agents", September 1–3, 2021, Bologna, Italy
f.sabbatini@unibo.it (F. Sabbatini); giovanni.ciatto@unibo.it (G. Ciatto); roberta.calegari@unibo.it (R. Calegari); andrea.omicini@unibo.it (A. Omicini)
http://federicosabbatini.apice.unibo.it (F. Sabbatini); https://about.me/gciatto (G. Ciatto); http://robertacalegari.apice.unibo.it (R. Calegari); http://andreaomicini.apice.unibo.it (A. Omicini)
0000-0002-0532-6777 (F. Sabbatini); 0000-0002-1841-8996 (G. Ciatto); 0000-0003-3794-2942 (R. Calegari); 0000-0002-6655-3869 (A. Omicini)

knowledge in a *sub-symbolic* way. Nevertheless, despite their sub-symbolic operation may prevent human users from understanding *how* they work, BB – and, in particular, ANN – are being increasingly applied to support forecasting and decision making in many different fields – including, but not limited to, marketing, customer/user profiling, social networks, predictive maintenance, etc. – because of their unprecedented predictive performance.

There exist, however, critical applications where black-box predictions or recommendations are unacceptable: for instance, healthcare, finance and law domains, or any other area of knowledge where decision making may affect critical aspects of human lives—e.g. health, wealth, freedom, etc. In all those cases, it is of paramount importance to rely on *explainable* predictions, recommendations, or suggestions, in order to let humans retain accountability and liability over the decision or choices they make.

Many strategies can be exploited to pursue the purpose of explainability [2]. Some authors suggest for instance to *only* rely on *interpretable* algorithms [3] – such as generalised linear models, decision trees, etc. – to obtain data-driven solutions that are explainable by construction. However, this may hinder predictive performance in the general case, as it essentially cuts off most effective algorithms—e.g., ANN. Another strategy consists of deriving *post-hoc* explanations [4], aimed at reverse-engineering the inner operation of a BB so as to make it explicit. In this way, data scientists can keep using prediction-effective algorithms such as ANN, while still attaining high predictive performance. The focus of this paper is on the latter strategy.

Symbolic knowledge extraction (SKE) is among the most promising means to derive *post-hoc* explanations for sub-symbolic predictors. Roughly speaking, the main idea behind SKE is to enable the construction of a *symbolic* surrogate model mimicking the behaviour of a given predictor. There, symbols may consist of intelligible knowledge, such as *rule* lists or trees. Such rules can then be exploited to either derive predictions or to better understand the behaviour of the original predictor.

SKE has been applied, for instance, to credit-risk evaluation [5, 6, 7], healthcare – i.e., to make early breast cancer prognosis predictions [8] and to help the diagnosis and discrimination among hepatobiliary disorders [9] or other diseases and dysfunctions [10] –, credit card screening [11], intrusion detection systems [12], and keyword extraction [13].

Despite the wide adoption of SKE, however, a unified and general-purpose software technology supporting it is currently lacking. In other words, the burden of implementing SKE algorithms is currently on data scientists alone, who are likely to realise custom solutions on a per-need basis. Other than producing inertia w.r.t. the adoption of SKE in modern data, such a lack of viable technologies is somewhat anachronistic in the data-driven AI era, where a plethora of libraries and frameworks are flourishing, targeting all major programming paradigms and platforms, and making state-of-the-art machine learning algorithms easily accessible to the general public—cf. SciKit-Learn[1] for Python, or Smile[2] for the Java Virtual Machine (JVM).

Accordingly, in this paper we present the design of PSyKE, a general-purpose Platform for Symbolic Knowledge Extraction aimed at filling the gap between the current state of the art of SKE and the available technology. More precisely, PSyKE is conceived as an open library where different sorts of knowledge extraction algorithms can be realised, exploited, or compared.

[1] https://scikit-learn.org/stable [Online; last accessed September 29, 2021].
[2] https://github.com/haifengl/smile [Online; last accessed September 29, 2021].

PSyKE supports rule extraction from both classifiers and regressors, and makes the extraction procedure as transparent as possible w.r.t. the underlying BB, depending on the particular extraction procedure at hand. Notably, it also supports the extraction of first-order logic (FOL) clauses, with the twofold advantage of providing human- and machine-interpretable rules as output. These can then be used as either an explanation for the original BB, or as a starting point for further symbolic computations. More precisely, the current implementation of PSyKE outputs logic programs [14, 15], expressed in Prolog syntax [16].

Furthermore, to demonstrate the versatility of PSyKE, we present a number of experiments involving rule extraction on a classification task performed on the Iris data set. In particular, we exemplify our framework against various BB predictors, and carry out a comparison between different extraction procedures applied to the same task. The comparison takes into account the fidelity of the extracted rules (w.r.t. the original BB) and their accuracy w.r.t. the data.

Accordingly, the remainder of this paper is organised as follows. Section 2 describes the state of the art for SKE as well as some background notion to fully understand the work. Section 3 presents the design of PSyKE, while in Section 4 some use cases showing how PSyKE can be exploited are reported. Conclusions are drawn in Section 5.

2. State of the Art

In this section we firstly overview the state of the art for symbolic knowledge extraction (Section 2.1). Then, we delve into the details of a selection of extraction algorithms—namely, the ones PSyKE implementation currently supports (Section 2.1.1–Section 2.1.3). The algorithm selection is performed by keeping *variety* (rather than exhaustivity) in mind, so as to demonstrate the operation and versatility of PSyKE. In particular, our aim is to exemplify the many application scenarios that a data scientist may meet—e.g., extraction from either classifiers or regressors, trained on either categorical or continuous data.

Finally, we briefly outline the currently-available software object-oriented frameworks for ML (Section 2.2). The overview is meant to make the paper self-contained, given that one of these frameworks provides PSyKE with pure ML functionalities—in particular, the design of PSyKE assumes basic classification or regression support to be available as a software library.

2.1. Knowledge Extraction

According to [17], a computational system is considered *interpretable* if human beings can easily understand its operation and outcomes. The majority of modern ML predictors, however, sacrifice interpretability to enhance the predictive performance, thus becoming increasingly complex. They do so by merely focusing on learning highly-predictive – yet *sub-symbolic* – input-output relations from data, while neglecting any attempt to make such relations *symbolic*, i.e., intelligible for human. For this reason, ML algorithms are often called *black boxes* [18].

To mitigate interpretability issues without sacrificing predictive performance, a number of authors from the XAI community have proposed means to produce *ex-post* explanations for sub-symbolic predictors—most notably, ANN and SVM. Explanations, in this case, consist of *surrogate* predictors trained to mimic the ones to be explained, as closely as possible.

In practice, among the manifold proposals, some authors describe methods to extract if-then-else rules [19, 20, 21], whereas others propose methods extracting decision trees [22]. While the shape of the extracted knowledge may vary from an extraction procedure to another, all the proposed methods share the trait of extracting *symbolic* (i.e. human-intelligible) knowledge out of *sub-symbolic* ML predictors. Given a trained predictor and a knowledge-extraction procedure applicable to it, the extracted rules/trees act as *explanations* for that predictor – or as a basis to build some –, provided that they retain high *fidelity* w.r.t. the underlying predictor [17]. The extracted knowledge may then enable further manipulations, such as merging the *know-how* of two or more BB models [23].

According to [24], knowledge extraction methods can be categorised along three orthogonal dimensions, namely: *(i)* the sort of learning tasks they support, *(ii)* the shape of the symbolic knowledge they produce, *(iii)* their *translucency*—i.e., the sort of BB algorithms they can extract symbols from.

About item *(i)*, one can distinguish among algorithms targeting classification tasks, regression tasks, or both. In other words, some extraction algorithms can only deal with BB classifiers – e.g. Rule-extraction-as-learning [19] (REAL, henceforth), TREPAN [22] and others [25, 26] –, while others can only deal with BB regressors – such as ITER [20], GridEx [21], REFANN [27], ANN-DT [28] and RN2 [29] –, and only a few can handle both—such as G-REX [30] and CART [31]. Notably, virtually all extraction methods proposed so far are tailored on *supervised* machine learning. To the best of our knowledge, no rule extraction procedure has been proposed targetting unsupervised or reinforcement learning tasks.

As far as item *(ii)* is concerned, decision rules [32, 33, 34] and trees [35, 36] are the most widespread human-understandable shapes for extracted knowledge, thus most methods produce one of these two structures. In both cases, decision rules or nodes are expressed in terms of the same input/output data types the original BB has been trained upon. So, for instance, an extraction procedure processing a BB classifier for N-dimensional numerical data, over K classes, will likely output rules/trees involving one or more *predicates* over N input variables $x_1, \ldots, x_n$ and K possible outcomes. In any case, however, extraction algorithms are further categorised w.r.t. the particular sort of predicates their output rules/trees may contain. Accordingly, conjunctions/disjunctions of inequality (e.g. $x_i \gtrless c$), or interval inclusion/exclusion expressions (e.g. $x_i \in [l, u]$) are commonly exploited for numerical data, while equality (e.g. $x_i = c$) or set-inclusion $x_i \in \{c_1, c_2, \ldots\}$ expressions may be exploited for categorical data. Finally, *M-of-N* rules are yet another possible choice in the case of boolean data.

The translucency dimension [37] from item *(iii)* refers to the need/capability of the extraction procedure to "look into" the *internal* structure of the underlying BB—i.e., to what extent it has to be taken into account during the extraction procedure. There are two major ways for categorising knowledge extractors w.r.t. translucency. During the extraction process, *decompositional* extractors may take into account the internal structure of the BB they operate upon, while *pedagogical* ones do not. For this reason, pedagogical approaches are usually more general – despite potentially less precise –, thus they can be applied to every BB predictor regardless of its kind, structure, and complexity.

The quality of knowledge-extraction procedures is evaluated through different indicators depending on the task to solve, for instance, fidelity and predictive performance measurements [38]. In particular, the former indicates how well the extracted knowledge mimics the underlying

Table 1
Summary of the knowledge-extraction algorithms supported by PSyKE.

Extraction Algorithm	Task	Translucency	Required Features	Knowledge Shape	Exhaustive
REAL	Classification	Pedagogical	One-hot encoded	Rule list	-
TREPAN	Classification	Pedagogical	One-hot encoded	Decision tree	✓
ITER	Regression	Pedagogical	Continuous	Rule list	-
GridEx	Regression	Pedagogical	Continuous	Rule list	✓
CART	Classification and regression	Pedagogical	Continuous or one-hot encoded	Decision tree	✓

black-box predictions, whereas the latter measures the explanator predictive power w.r.t. the data. In all cases, measurements should be taken via the same scoring function used for assessing the BB performance—which in turn depends on the task it performs. In the particular case of black-box classifiers, examples of performance measurements are accuracy, precision, recall, and F1-score; for BB regressors, the mean absolute/squared error (MAE/MSE) and the R^2 scores could be exploited.

In the following, we provide a more detailed description of some extraction procedures currently supported in the PSyKE framework, grouped by the task they address—i.e. classification, regression, or both of them. All the algorithms are *pedagogical*, thus they only rely on the BB inputs and outputs, and do not inspect the inner structure of the underlying BB. This is why they can be applied to any kind of arbitrarily-complex BB. In any case, structured data are required. The same algorithms are summarised in Table 1.

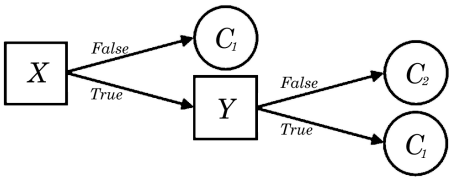
2.1.1. Extraction from Classifiers

Rule-extraction-as-learning REAL [19] is a pedagogical algorithm to extract conjunctive rules from trained BB classifiers by using a learning process driven by sampling and queries. Output rules can be either *if-then* or *M-of-N* rules. An example of *if-then* output rule is the following: *Output class is* C *if* $\{X, Y, Z\}$ *are True and* $\{U, V\}$ *are False*, where *U, V, X, Y, Z* are one-hot encoded input features. Such features are *True* if they are 1, *False* otherwise.

Output rules are disjunctive normal form expressions; each term is the conjunction of a data set feature subset, adequately generalised by dropping each non-discriminant antecedent. REAL cannot handle real-valued features, but only binary ones.

TREPAN TREPAN [22] is a pedagogical algorithm able to extract symbolic and comprehensible model representations from trained classifier BB by inducing a decision tree approximating the BB represented concept. It usually maintains high fidelity levels w.r.t. the underlying BB while being comprehensible and accurate. It is general in its applicability and well scalable with complex models or problems. However, as REAL, it cannot be applied to real-valued features.

In Figure 1 an example of output tree is reported. Internal nodes are represented by squares, while leaves are circles. Variables reported inside the internal nodes are the split criteria for the subtree creation. C_i are the class labels corresponding to each leaf.



Output class is C_1 if $\begin{cases} (X \text{ is } \textit{True}) \text{ or} \\ (X \text{ is } \textit{False} \text{ and } Y \text{ is } \textit{True}) \end{cases}$

Output class is C_2 otherwise.

Figure 1: Example of TREPAN output tree (left) and corresponding rules (right).

2.1.2. Extraction from Regressors

ITER ITER [20] is a pedagogical algorithm for building predictive rules from trained BB regressors of any kind. Its main idea is to iteratively expand a number of hypercubes until they cover the whole input space. Each of them is finally converted into an *if-then* rule of the following format: *Output constant is* C *if* $X_1 \in [l_1, u_1]$ *and ... and* $X_k \in [l_k, u_k]$, where l_i and u_i are the lower-bound and the upper-bound for variable X_i. There, the preconditions of the rule describe a k-dimensional hypercube. Indeed, ITER supports continuous input features, differently from the other algorithms presented so far.

GridEx GridEx [21] is another pedagogical extraction algorithm for regressors; it is an extension of ITER aimed at overcoming its major drawback: non-exhaustivity. GridEx adopts a top-down approach to iteratively partition the input feature space in a user-defined number of hypercubes or in an automatic way accordingly to a user-defined strategy based on feature importance. As ITER, GridEx produces *if-then* rules and only accepts data sets with real-valued input features, but it is always exhaustive by design. Since the procedure associates each hypercube to a rule, a merging phase is performed after every iteration as an optimisation to reduce the number of rules.

2.1.3. General-Purpose Extractors

CART CART [31] is an algorithm for building decision trees that can be used to face both classification and regression tasks. CART is not properly a knowledge-extraction procedure, but from its output tree it is straightforward to obtain a rule tree, since each node of the CART tree corresponds to a constraint on a certain feature and thus each path from the root to a leaf is a single complete classification or regression rule. The algorithm can be summarised via the following instructions: *(i)* initialise the tree root node; *(ii)* find optimal splits and add new internal nodes and leaves accordingly; *(iii)* stop the algorithm based on one or more criteria—e.g., leaf number or tree depth. Pruning algorithms can be applied to reduce the number of leaves.

2.2. Object-Oriented Programming Frameworks for ML

In order to make ML solutions easily available, a number of frameworks – especially exploiting object-oriented programming (OOP) – have been developed. Such frameworks usually provide users with powerful abstractions for modelling BB models as well as for performing data set pre-processing, feature engineering and predictive performance measurements. Among the most supported ML models, there are ANN, SVM, and decision trees for both classification and

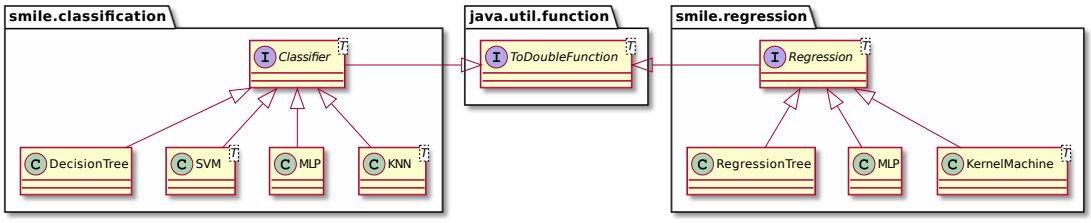


Figure 2: Overview on the API of the Smile library supporting classification and regression tasks.

regression tasks. The most complete frameworks also provide utilities packages to ease the data set reading from (and writing into) files and to perform feature selection, beyond many other tools for natural language processing, linear algebra, and data visualisation. Examples of state-of-the-art OOP frameworks for ML are SciKit-Learn [39] for Python and Smile for the JVM. We adopted the latter for the design and development of PSyKE because of its JVM support.

Smile (Statistical Machine Intelligence and Learning Engine) is defined as "a fast and comprehensive machine learning engine"[3] for Java, Scala, and Kotlin. It is worthwhile to notice that the package provides data types for easily managing data set, feature vectors, and tuples (intended as data set columns and rows, respectively) other than all the aforementioned tools.

Figure 2 depicts a (partial) UML class diagram representing the major interfaces and classes composing Smile's supervised learning API. Package names are explicitly indicated to avoid confusion between homonymous classes. Notably, each kind of ML predictor has a dedicated class and each class implements either the `Classifier` or the `Regression` interface. Both interfaces descend from the `ToDoubleFunction` interface, which is, therefore, the most adequate type to represent any supervised ML predictor. This design lets developers easily build more complex concepts over the packages offered by Smile—as in the PSyKE prototype. Of course, the same design could be replicated on different platforms (e.g. Python) and libraries (e.g. SciKit-Learn)—as long as they provide similar API to train and use ML classifiers or regressors.

3. PSyKE

PSyKE is a software library providing general-purpose support to the extraction of logic rules out of BB predictors by letting users choose the most adequate extraction method for the task and data at hand. PSyKE exposes a unified API covering virtually all extraction algorithms targeting supervised learning tasks. Currently, the implementation of PSyKE involves several interoperable, interchangeable, and comparable extraction procedures – namely, the ones mentioned in Section 2.1 –, granting access to state-of-the-art knowledge-extraction algorithms to both researchers and data scientists. PSyKE is conceived as an open-ended project, which can be exploited to design and implement new extraction procedures behind a unique API.

Essentially, PSyKE is designed around the notion of *extractor*, whose API is depicted in Figure 3. Within the scope of PSyKE, an extractor is any algorithm accepting a ML predictor – either a classifier or a regressor – as input, and producing a *theory* of *logic* rules as output.

[3] https://haifengl.github.io [Online; last accessed September 29, 2021].

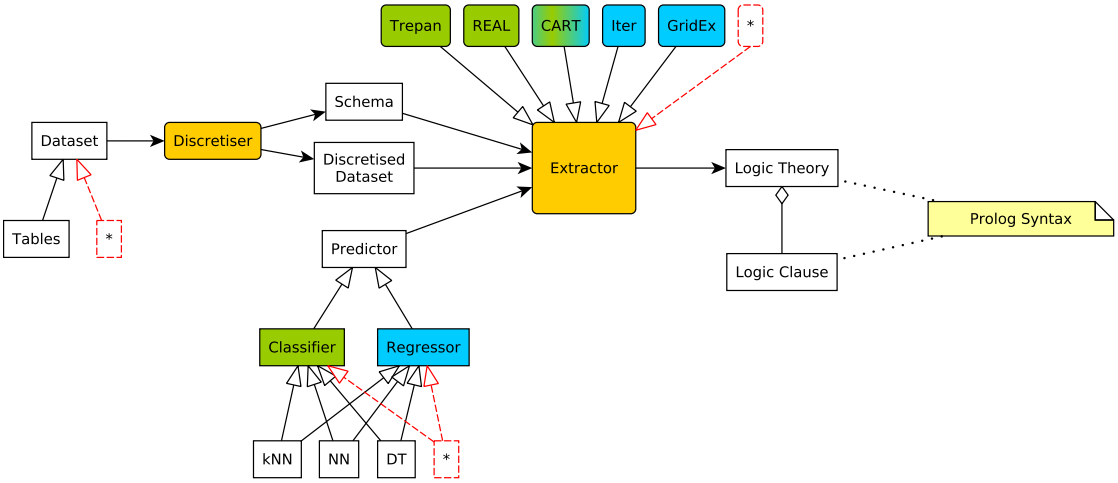


Figure 3: PSyKE design.

To perform their job, PSyKE extractors require additional information about the data set the input predictor has been trained upon. In the general case, such information consists of the data set itself and its schema—i.e., a formal description of the names and the data types of all features characterising the data set itself. More precisely, data sets are required to let extraction procedures inspect BB behaviour – and therefore build the corresponding output rules –, whereas schemas are required to let *(i)* the extraction procedure take informed decisions on the basis of the feature *types*, *(ii)* the extracted knowledge be clearer by referring to the feature *names*. For all these reasons, extractors expect a data set and its schema metadata to be provided in input as well.

Many extraction procedures can operate on discrete/binary data only. This is commonly made necessary by the shape of the extracted rules—which consists of simple predicative statements about some feature value. However, it is also very common in data science to meet data sets involving continuous attributes as well. Accordingly, extracting rules out of predictors trained on continuous data may be troublesome in the general case. To circumvent this issue, PSyKE also provides some facilities aimed at discretising (or binarising) data sets including continuous (or categorical) data. When these are in place, extractors should be provided with the discretised/binarised schema as well, to be able to produce the clearest rules possible.

Accordingly, in the rest of this section we detail *(i)* the general design of the PSyKE library and API, *(ii)* the one-hot encoding facilities, *(iii)* the general shape of the extracted logic theory.

3.1. General API

As depicted in Figure 4, a pivotal role in the design of PSyKE is played by the `Extractor` interface – reported in Figure 4 –, defining the general contract of any knowledge-extraction procedure. More precisely, it leverages on the notions of `DataFrame` and `Theory`, borrowed from Smile and 2P-Kt [40, 41], respectively. All the PSyKE extractors expose *(i)* a method for extracting an explainable theory from a BB model and *(ii)* a method to make predictions by

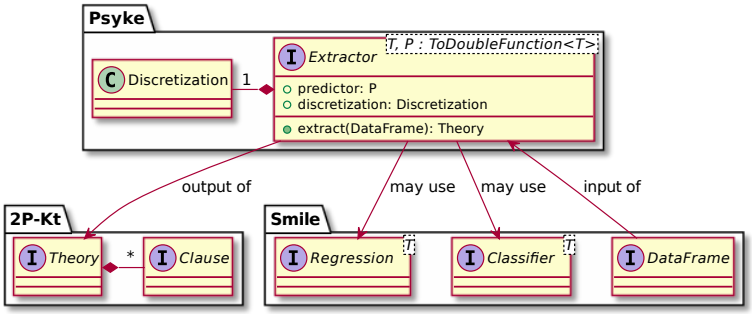


Figure 4: PSyKE's `Extractor` interface

using the extracted rules. The common API makes it possible to switch between different PSyKE extractors with no need of major changes in the code.

3.2. Discretisation

A large number of the knowledge-extraction procedures – in the same way as many ML algorithms – require either a discrete or binary input space—i.e. all input features must be either categorical or one-hot encoded, respectively. For instance, REAL and TREPAN require exclusively one-hot encoded data, whereas ITER and GridEx require continuous data. CART can accept both continuous and one-hot encoded features, but not categorical ones. Unfortunately, most real-world applications are described by real-valued variables and measurements, thus making the application of such algorithms impractical. The general way to overcome this limitation is to rely on some discretisation/binarisation method among the many available in the literature—e.g., [42, 43, 44, 45, 46, 47, 48].

Briefly speaking, discretisation is the process of transforming a datum from some continuous space $I \subseteq \mathbb{R}$ into a discrete space $\{I_1, \ldots, I_n\}$ such that $\forall i, j = 1, \ldots, n$: $I_i \subset I \wedge I \equiv \bigcup_i I_i \wedge i \neq j \Leftrightarrow I_i \cap I_j = \varnothing \wedge i < j \Leftrightarrow \forall x \in I_i, \forall y \in I_j : x < y$. Similarly, binarisation (a.k.a. one-hot encoding) is the process of transforming a datum from some discrete space $X = \{x_1, \ldots, x_n\}$ into a binary space $B = \{b_1, \ldots, b_n\}$ where for each $i = 1, \ldots, n$: b_i is 1 if the datum is equal to x_i, 0 otherwise. Of course, these methods imply a considerable increase in the dimensionality of a data set—e.g., one-hot encoding makes categorical attributes with 4 distinct values be converted into 4 different boolean features. This is far from being an issue: in some cases, it is possible to achieve even better classification performances by using discretised attributes rather than continuous, as demonstrated in [49].

PSyKE provides different procedures to manipulate input features: *(i)* a discretisation for continuous features, mapping real intervals into categorical features, and *(ii)* a one-hot encoding for categorical features, mapping exact values to boolean features. Notably, PSyKE traces the input feature transformations by creating a data structure that associates the initial name of the attribute and the newly created features with the corresponding constraints. An example of PSyKE binarisation and corresponding output data structure is reported in Figure 5. This example considers the *petal length* attribute of the Iris data set and adopts the default supervised discretisation method available in our framework. The initial continuous feature values are

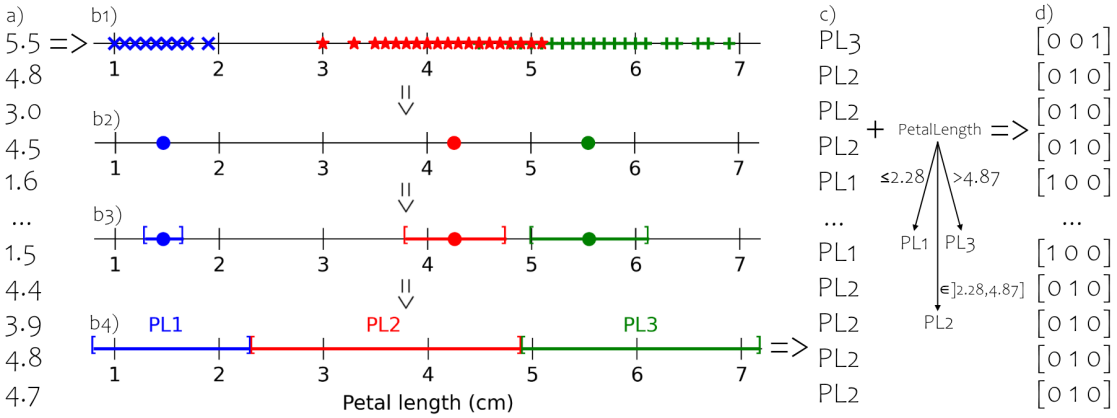


Figure 5: PSyKE discretisation and binarisation procedure.

labelled in Figure 5 with a), and graphically represented in the b1) plot. PSyKE discretisation algorithm consists in calculating the mean attribute value and the corresponding standard deviation *for each data set class*. An interval is then initialised for each class, with lower and upper bounds equal to the mean value—cf. plot b2). Each interval is iteratively expanded until convergence—i.e., when the whole feature space is covered without overlapping intervals. To achieve this, during every iteration all the intervals are symmetrically expanded of a value equal to the corresponding standard deviation – in each direction, as in plot b3) –, in order to create intervals with adaptive size. An expansion is inhibited when *(i)* the lower (upper) bound of an interval exceeds the minimum (maximum) value of the feature space, or *(ii)* two adjacent intervals are overlapping. In the first case, the feature minimum (maximum) value is taken as the final interval lower (upper) bound. In the second case, the overlapping intervals are only expanded up to the mean value between their respective boundaries. When all intervals have been calculated – cf. plot b4) – the continuous attribute values are converted accordingly into categorical values. The discretised output values are labelled in the example with c). In this step PSyKE also produces a data structure for keeping the discretisation details, to be able to produce more compact rules during the extraction procedures. The last step – labelled with d) – is the one-hot encoding of the discrete values into arrays of binary data—i.e., the unique format accepted by several extraction procedures, such as REAL and TREPAN.

3.3. Output rules

PSyKE extractors output knowledge in the form of logic theories – i.e. lists of Horn clauses –, notably in Prolog syntax. We choose the Prolog syntax to make them simultaneously interpretable by both humans and machines. More precisely, PSyKE output theories are structured as lists of Prolog rules or facts. Rule heads are $(n + 1)$-ary predicates, where n is the number of input features in the dataset. These predicates carry n variables – i.e., one for each input feature – and either a constant or a list – i.e., the output value(s) – as argument. Predicate names recall the classification/regression under study. Without lack of generality, in the following we assume the case under study to involve *mono-dimensional* classification/regression tasks.

Rule bodies can be empty – if rules are *facts* – or conjunctions of literals where each literal is a predicate expressing inequality, equality, or interval inclusion between attribute actual values and fixed constants calculated through the extraction process. Accordingly, a rule-extraction procedure targeting a mono-dimensional classification or regression $task$ on a data set having n input features and m relevant output values, shall output theories of the following form:

$$\begin{array}{rcl} \langle task \rangle(\mathtt{x}_1, \ldots, \mathtt{x}_n, \mathtt{y}_1) & \texttt{:-} & p_{1,1}(\bar{\mathtt{x}}),\ \ldots,\ p_{n,1}(\bar{\mathtt{x}}). \\ \langle task \rangle(\mathtt{x}_1, \ldots, \mathtt{x}_n, \mathtt{y}_2) & \texttt{:-} & p_{1,2}(\bar{\mathtt{x}}),\ \ldots,\ p_{n,2}(\bar{\mathtt{x}}). \\ & \vdots & \\ \langle task \rangle(\mathtt{x}_1, \ldots, \mathtt{x}_n, \mathtt{y}_m) & \texttt{:-} & p_{1,m}(\bar{\mathtt{x}}),\ \ldots,\ p_{n,m}(\bar{\mathtt{x}}). \end{array}$$

where *(i)* $task$ is the $(n+1)$-ary relation representing the classification or regression task at hand, *(ii)* each $\mathtt{x}_i$ is a logic variable named after the i^{th} input attribute of the currently available data set, *(iii)* $\bar{\mathtt{x}}$ is the n-nuple $\mathtt{x}_1, \ldots, \mathtt{x}_n$, and *(iv)* each $p_{i,j}$ is either a n-ary predicate expressing some constraint about one, two or more variables, or the `true` literal—which can be omitted. Notice that, in classification tasks, the total amount of rules (m) may still be greater than the total amount of classes (k), as there may be more than one rule for the same class.

Currently, the supported sorts of predicates in rules bodies – i.e. the admissible shapes for each $p_{i,j}$ – are as follows:

equality involving a single variable and a constant—e.g. $\mathtt{x} = c$, where c is a constant of any sort (possibly, a number)[4]

inequalities involving a single variable and a constant—e.g. $\mathtt{x} \gtrless c$

interval inclusion involving a single variable and two constants—e.g. `x in` $[l, u]$, where $l, u \in \mathbb{R}$ and $l < u$

interval exclusion like the above, but negated—e.g. `x not_in` $[l, u]$

M-of-N involving N variables—e.g. `at_least`$(M, [\mathtt{x}_1, \ldots, \mathtt{x}_N])$, where $M, N \in \mathbb{N}_{\geq 0}$

Despite many other forms can be adopted for the output theories, we argue the proposed one is a good trade-off among human and machine interpretability. In fact, rules of this form are well-formed logic programs, which may be executed by a logic reasoner—such as a Prolog interpreter. Furthermore, the proposed form is open to many sorts of post-processing. For instance, recurrent conjunctions of predicates in rules bodies may be factorised into their own general-purpose rules. Similarly, redundant or cumbersome sub-expressions may be simplified.

However, the proposed form is far from perfection: we plan to explore alternative directions in the future. Noticeably, using Prolog syntax does not impose exploiting also its semantics—i.e., different interpreters can be exploited over such Prolog rules. For instance, on the one side, by exploiting a Prolog interpreter, rules are interpreted as functional (one-way)—meaning that it is possible to compute a prediction given an assignment of all input variables, but it is not possible to generate a correct assignment of those input variables given the expected prediction alone. On the other side, a Constraint Logic Programming solver [50, 51] may interpret the same rules as constraints, and compute coherent assignments for any subset of both input and output variables—providing rules with a relational (two-ways, generative) semantics.

[4] The same result could be attained by allowing constants in rules heads.

4. Case Study: The Iris Data Set

Here we exemplify the effectiveness and versatility of PSyKE by describing its exploitation in a toy scenario. In particular we exploit PSyKE to extract Prolog rules on a number of classifiers trained on the well-known Iris data set.[5] Notably, the Iris data set contains 150 rows describing as many individuals of the Iris flower. For each exemplary, 4 continuous input features – petal and sepal width and length – are recorded, other than a categorical class label—i.e. which particular sort of Iris plant the exemplary has been classified as. There are three particular sub-sorts of Iris in this data set – namely, Setosa, Virginica, and Versicolor –, and the 150 examples are evenly distributed among them—i.e. there are 50 instances for each class.

The experimental setting is as follows. First, we train 3 different sorts of classifiers on the Iris data set—namely, a k-nearest-neighbors (kNN), a multi-layer perceptron (MLP), and a decision tree (DT). Then we let PSyKE extract logic rules out of these classifiers using as many extraction procedures. In particular, we rely on REAL, Trepan, and Cart. A portion (50%) of the original data set – namely, the test set – is put aside *before* training to later enable the evaluation of the extracted rule predictive performance.

Accordingly, within the scope of this experiment, we rely on *accuracy* as the preferred metric for both predictive performance and fidelity—where the former measures how good a classifier or the corresponding extracted rules are in classifying Iris instances in absolute terms, while the latter measures the adherence of the extractor output rules w.r.t. the original classifier.

4.1. The experiment

Let us assume the Iris data set can be loaded from a CSV file via a Kotlin script using Smile. The Iris data set only contains continuous features. Therefore, Cart is the only algorithm that can be *directly* applied to it, whereas REAL and Trepan can only operate on binary data. Accordingly, PSyKE provides a simple two-step procedure to binarise the data, involving both discretisation and one-hot encoding: a data set can be discretised, one-hot encoded, and split into training and test set via a couple of instructions, providing the percentage of samples to be taken apart from the whole data set to attain the test set. As the next step, we train 3 different classifiers on the training set—namely a kNN, a MLP, and a DT.[6] Finally, in the following paragraphs, we show how rules can actually be extracted and what their ultimate shape actually is.

4.1.1. REAL

The PSyKE REAL algorithm can be applied to any Smile `Classifier` model parametrised with a `DoubleArray`—e.g., the aforementioned MLP and kNN are suitable, whereas the DT is not. The extracted theory is dependent from the training set; different training instances can produce different rules, resulting in slight variations also in the output theory complexity—intended as number of clauses and terms. An example is reported in the following:

[5] https://archive.ics.uci.edu/ml/datasets/iris [Online; last accessed September 29, 2021].

[6] Code of experiments is available at https://github.com/sabbatinif/PSyKE. Please consider reading the documentation of Smile for a more detailed explanation of the model-specific parameters: https://haifengl.github.io/classification.html and https://haifengl.github.io/regression.html.

```
iris(SepalLength, SepalWidth, PetalLength, PetalWidth, setosa) :-
  PetalWidth =< 0.65.
iris(SepalLength, SepalWidth, PetalLength, PetalWidth, versicolor) :-
  PetalWidth in [0.65, 1.64].
iris(SepalLength, SepalWidth, PetalLength, PetalWidth, virginica) :-
  PetalWidth > 1.64.
```

The theory produces an input space partitioning as reported in Figure 6c. It is worthwhile to notice that – especially with more complex data sets – the partitioning could be non-exhaustive—i.e., the logic rules could be unable to classify some samples.

4.1.2. TREPAN

PSyKE provides a TREPAN algorithm applicable under the same constraint described above for REAL and also its output rules can vary with different training sets. Differently from REAL, TREPAN accepts as input 3 optional parameters stating the minimum number of samples to consider for performing further splits (`minExamples`, default: 0), the maximum depth of the produced tree (`maxDepth`, default: 0, i.e. no constraints), and the criterion to adopt for the best split selection (`splitLogic`). A default logic is chosen if not otherwise specified. An example of output theory is reported in the following:

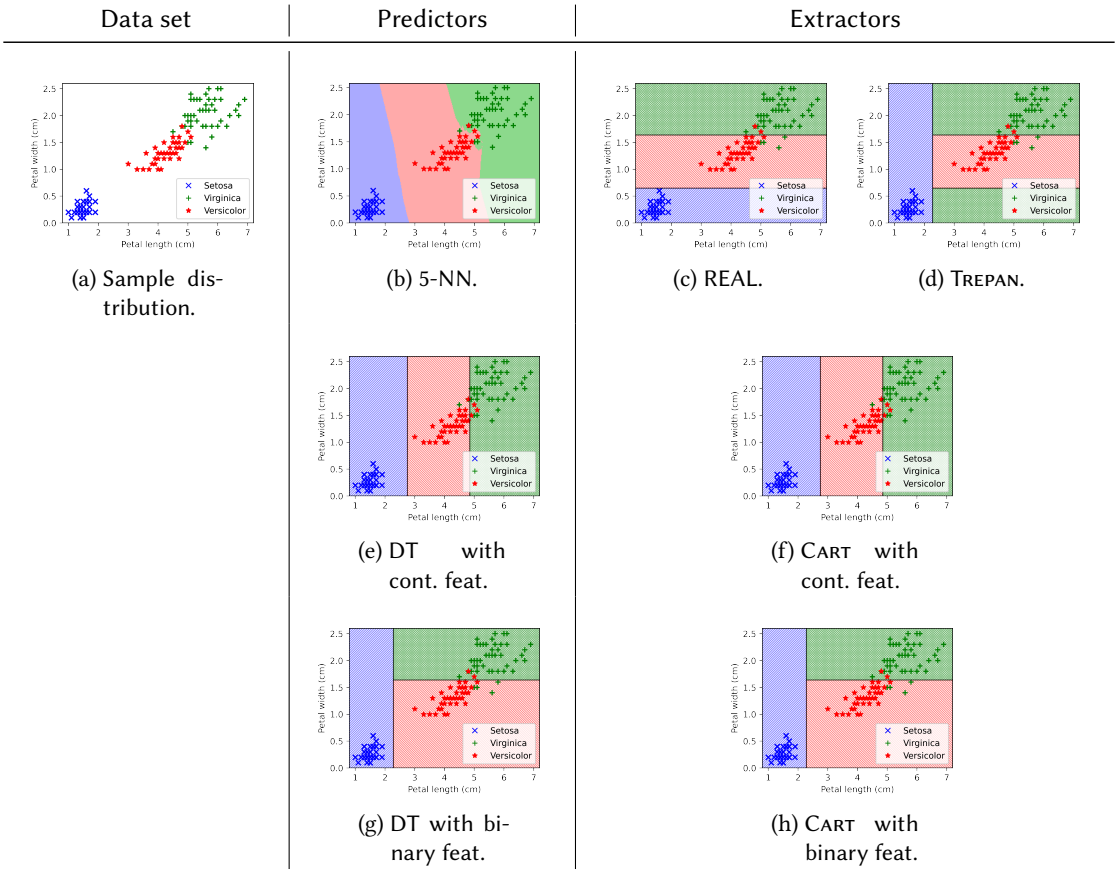
```
iris(SepalLength, SepalWidth, PetalLength, PetalWidth, setosa) :-
  PetalLength =< 2.28.
iris(SepalLength, SepalWidth, PetalLength, PetalWidth, virginica) :-
  PetalLength > 2.28, PetalWidth not_in [0.65, 1.64].
iris(SepalLength, SepalWidth, PetalLength, PetalWidth, versicolor) :- true.
```

This theory produces an input space partitioning as reported in Figure 6d. In this case, the partitioning is always exhaustive.

4.1.3. CART

CART is the third algorithm included in PSyKE to tackle classification tasks. It is the only procedure applicable to a `DecisionTree` classifier and it does not require any extra parameter, since the extraction only relies on the tree structure of the `DecisionTree`—that is, all the parameters have to be tuned during the `DecisionTree` creation and training. The `DecisionTree` can accept both one-hot encoded and continuous features. In the same way as many other algorithms, CART is able to achieve comparable or even better results when relying on a good discretisation/one-hot encoding technique. In the following an example of theory obtained with continuous features is reported:

```
iris(SepalLength, SepalWidth, PetalLength, PetalWidth, setosa) :-
  PetalLength =< 2.75.
iris(SepalLength, SepalWidth, PetalLength, PetalWidth, versicolor) :-
  PetalLength > 2.75, PetalLength =< 4.85.
iris(SepalLength, SepalWidth, PetalLength, PetalWidth, virginica) :-
  PetalLength > 2.75, PetalLength > 4.85.
```



(a) Sample distribution. (b) 5-NN. (c) REAL. (d) TREPAN. (e) DT with cont. feat. (f) CART with cont. feat. (g) DT with binary feat. (h) CART with binary feat.

Figure 6: Comparison between Iris data set input space partitionings performed by the algorithms implemented in PSyKE. Only the two most relevant features are reported—i.e., petal width and length.

Figure 6f reports the corresponding input space partitioning. The output rules are always exhaustive. The same data set, but previously one-hot encoded, leads to the following theory and to the partitioning reported in Figure 6h:

```
iris(SepalLength, SepalWidth, PetalLength, PetalWidth, versicolor) :-
  PetalLength > 2.28, PetalWidth =< 1.64.
iris(SepalLength, SepalWidth, PetalLength, PetalWidth, virginica) :-
  PetalLength > 2.28, PetalWidth > 1.64.
iris(SepalLength, SepalWidth, PetalLength, PetalWidth, setosa) :-
  PetalLength =< 2.28.
```

4.2. Results

In the following we report the results of REAL, TREPAN, and CART applied to the Iris data set. All the results are resumed in Figure 6. Figure 6a reports the Iris data set sample distribution in the input space, with emphasis only on the two most relevant features—i.e., petal width and length. Column *Predictors* represents the ML step of the process. Accordingly, Figure 6b,

Table 2
Comparison between accuracy and fidelity measurements with different combinations of extractor algorithms and underlying models.

Predictor		Extractor		
Type	Accuracy	Algorithm	Fidelity	Accuracy
5-NN	0.94	REAL	0.98	0.95
		Trepan	0.96	0.96
MLP	0.92	REAL	0.95	0.95
		Trepan	0.99	0.92
DT (continuous features)	0.92	Cart	1.00	0.92
DT (binarised features)	0.96	Cart	1.00	0.96

Figure 6e and Figure 6g represents the decision boundaries of a 5-NN predictor and 2 decision trees trained with a half of the data set samples, respectively. The first DT is trained with the original Iris data set; the other with the binarised version. Finally, column *Extractors* represents the PSyKE output. In particular, different extraction procedures – namely, REAL and Trepan – applied to the 5-NN are depicted in Figure 6c and Figure 6d, respectively – and Cart extraction applied to decision trees is depicted in Figure 6f and Figure 6h.

It is worth noticing that *(i)* Cart – for its design – always produces partitionings equivalent to those of the underlying DT, and *(ii)* all extractor output partitionings are different – namely, a procedure only uses the petal width attribute, another one uses only the petal length and the remaining use both input features –, but all solutions share a similar predictive performance.

Omitted in Figure 6, the results of REAL and Trepan applied to a MLP are presented in Table 2, where a numerical assessment of the aforementioned predictors and extractors is reported. Values are averaged upon 25 executions, each one with different random train/test partitionings, but same test set percentage and same parameters for predictors and extractors. The table reports the underlying predictor accuracy as well as the fidelity and accuracy of the extraction procedure. We plan to enhance comparisons between different extractors through fidelity assessments carried out by measuring the decision boundary overlapping regions. From Table 2 it can be noticed that the Cart extractor always has a fidelity of 1.0, since it only inspects the underlying decision tree nodes to build its output rules without any knowledge loss. This implies that PSyKE Cart is an equivalent but explainable alternative to Smile `DecisionTree` model, always producing the same output predictions. As for the other extractors, both REAL and Trepan are able to achieve good results in terms of fidelity and accuracy – always above 0.9 in our experiments – in some cases even with a better performance w.r.t. the original model.

5. Conclusions

In this paper we present the design of PSyKE, a new general-purpose platform supporting symbolic knowledge extraction from opaque ML predictors. PSyKE offers many comparable and interchangeable extraction procedures providing as output first-order logic clauses. It can be

exploited in the majority of supervised learning tasks—i.e., classification and regression tasks.

In the future we plan to enrich PSyKE with other state-of-the-art extraction algorithms, comparison metrics between the implemented procedures, and other utilities—i.e., discretisation strategies. We also plan to explore other formalisms to present output rules – e.g. the ProbLog syntax to introduce the concept of probabilistic rule –, as well as other representations and extraction procedures which are better suited to manage data sets involving a wide number of features.

From a research perspective, we aim at further investigating the effectiveness of PSyKE in running EU projects, like StairwAI[7] and Expectation [52].

StairwAI is an H2020 project aimed at providing a service layer for the AI-on-demand platform,[8] with the purpose of aiding both individual and companies to *(i)* find the most adequate AI asset for their needs – requiring mapping of use cases to proper AI assets –, and *(ii)* experimenting selected AI assets on custom data and on specific problems, using the platform itself—thus requiring tools for predicting the hardware resources needed for running the corresponding software.

Expectation is a CHIST-ERA IV project[9] aimed at exploring the provisioning of *personalised* explanations for ML techniques by combining SKE and multi-agent-based negotiation and argumentation. There, symbolic knowledge is expected to act as the *lingua franca* among many heterogeneous ML-based predictors – possibly trained on different data sets, via different algorithms, at different locations –, hosted by as many software agents. Personalisation and predictive accuracy are therefore attained by combining the symbolic knowledge extracted by several agents. The combination takes advantage of negotiation and argumentation techniques, possibly involving the users themselves.

Both projects massively rely on sub-symbolic AI, and in both cases the need of making sub-symbolic knowledge explainable is prominent. PSyKE could then be applied to extract logic rules and reveal information about the path that leads to a certain prediction—in the explanation perspective. Since PSyKE currently works as a distiller of knowledge, further investigation will be devoted to the explanation of a single outcome (prediction of a model). Moreover, it could be interesting to compare results with those obtained by directly learning a symbolic model .

Acknowledgments

This paper has been partially supported by *(i)* the European Union's Horizon 2020 research and innovation programme under G.A. no. 101017142 (StairwAI project), and by *(ii)* the CHIST-ERA IV project CHIST-ERA-19-XAI-005, co-funded by the EU and the Italian MUR (Ministry for University and Research).

[7] https://cordis.europa.eu/project/id/101017142 [Online; last accessed September 29, 2021].
[8] https://cordis.europa.eu/project/id/825619 [Online; last accessed September 29, 2021].
[9] https://www.chistera.eu/projects/expectation [Online; last accessed September 29, 2021].

References

[1] A. Rocha, J. P. Papa, L. A. A. Meira, How far do we get using machine learning black-boxes?, International Journal of Pattern Recognition and Artificial Intelligence 26 (2012) 1261001–(1–23). doi:10.1142/S0218001412610010.

[2] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, D. Pedreschi, A survey of methods for explaining black box models, ACM Computing Surveys 51 (2018) 1–42. doi:10.1145/3236009.

[3] C. Rudin, Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead, Nature Machine Intelligence 1 (2019) 206–215. doi:10.1038/s42256-019-0048-x.

[4] E. M. Kenny, C. Ford, M. Quinn, M. T. Keane, Explaining black-box classifiers using post-hoc explanations-by-example: The effect of explanations and error-rates in XAI user studies, Artificial Intelligence 294 (2021) 103459. doi:10.1016/j.artint.2021.103459.

[5] B. Baesens, R. Setiono, C. Mues, J. Vanthienen, Using neural network rule extraction and decision tables for credit-risk evaluation, Management Science 49 (2003) 312–329. doi:10.1287/mnsc.49.3.312.12739.

[6] B. Baesens, R. Setiono, V. De Lille, S. Viaene, J. Vanthienen, Building credit-risk evaluation expert systems using neural network rule extraction and decision tables, in: V. C. Storey, S. Sarkar, J. I. DeGross (Eds.), ICIS 2001 Proceedings, Association for Information Systems, 2001, pp. 159–168. URL: http://aisel.aisnet.org/icis2001/20.

[7] M. T. A. Steiner, P. J. Steiner Neto, N. Y. Soma, T. Shimizu, J. C. Nievola, Using neural network rule extraction for credit-risk evaluation, International Journal of Computer Science and Network Security 6 (2006) 6–16. URL: http://paper.ijcsns.org/07_book/200605/200605A02.pdf.

[8] L. Franco, J. L. Subirats, I. Molina, E. Alba, J. M. Jerez, Early breast cancer prognosis prediction and rule extraction using a new constructive neural network algorithm, in: Computational and Ambient Intelligence (IWANN 2007), volume 4507 of *LNCS*, Springer, 2007, pp. 1004–1011. doi:0.1007/978-3-540-73007-1_121.

[9] Y. Hayashi, R. Setiono, K. Yoshida, A comparison between two neural network rule extraction techniques for the diagnosis of hepatobiliary disorders, Artificial intelligence in Medicine 20 (2000) 205–216. doi:10.1016/s0933-3657(00)00064-6.

[10] G. Bologna, C. Pellegrini, Three medical examples in neural network rule extraction, Physica Medica 13 (1997) 183–187. URL: https://archive-ouverte.unige.ch/unige:121360.

[11] R. Setiono, B. Baesens, C. Mues, Rule extraction from minimal neural networks for credit card screening, International Journal of Neural Systems 21 (2011) 265–276. doi:10.1142/S0129065711002821.

[12] A. Hofmann, C. Schmitz, B. Sick, Rule extraction from neural networks for intrusion detection in computer networks, in: 2003 IEEE International Conference on Systems, Man and Cybernetics, volume 2, IEEE, 2003, pp. 1259–1265. doi:10.1109/ICSMC.2003.1244584.

[13] A. Azcarraga, M. D. Liu, R. Setiono, Keyword extraction using backpropagation neural networks and rule extraction, in: The 2012 International Joint Conference on Neural Networks (IJCNN 2012), IEEE, 2012, pp. 1–7. doi:10.1109/IJCNN.2012.6252618.

[14] J. W. Lloyd (Ed.), Computational Logic, Springer, 1990. doi:10.1007/978-3-642-76274-1.

[15] G. Metakides, A. Nerode, Principles of logic and logic programming, volume 13 of *Studies in Computer Science and Artificial Intelligence*, Elsevier, 1996. URL: https://www.elsevier.com/books/principles-of-logic-and-logic-programming/metakides/978-0-444-81644-3.

[16] A. Colmerauer, P. Roussel, The birth of Prolog, in: History of programming languages—II, ACM, 1996, pp. 331–367. doi:10.1145/234286.1057820.

[17] G. Ciatto, D. Calvaresi, M. I. Schumacher, A. Omicini, An abstract framework for agent-based explanations in AI, in: A. El Fallah Seghrouchni, G. Sukthankar, B. An, N. Yorke-Smith (Eds.), 19th International Conference on Autonomous Agents and MultiAgent Systems, IFAAMAS, 2020, pp. 1816–1818. URL: http://ifaamas.org/Proceedings/aamas2020/pdfs/p1816.pdf.

[18] Z. C. Lipton, The mythos of model interpretability, Queue 16 (2018) 31–57. doi:10.1145/3236386.3241340.

[19] M. W. Craven, J. W. Shavlik, Using sampling and queries to extract rules from trained neural networks, in: Machine Learning Proceedings 1994, Elsevier, 1994, pp. 37–45. doi:10.1016/B978-1-55860-335-6.50013-1.

[20] J. Huysmans, B. Baesens, J. Vanthienen, ITER: An algorithm for predictive regression rule extraction, in: Data Warehousing and Knowledge Discovery (DaWaK 2006), Springer, 2006, pp. 270–279. doi:10.1007/11823728_26.

[21] F. Sabbatini, G. Ciatto, A. Omicini, GridEx: An algorithm for knowledge extraction from black-box regressors, in: D. Calvaresi, A. Najjar, M. Winikoff, K. Främling (Eds.), Explainable and Transparent AI and Multi-Agent Systems. Third International Workshop, EXTRAAMAS 2021, Virtual Event, May 3–7, 2021, Revised Selected Papers, volume 12688 of *LNCS*, Springer Nature, Basel, Switzerland, 2021, pp. 18–38. doi:10.1007/978-3-030-82017-6_2.

[22] M. W. Craven, J. W. Shavlik, Extracting tree-structured representations of trained networks, in: D. S. Touretzky, M. C. Mozer, M. E. Hasselmo (Eds.), Advances in Neural Information Processing Systems 8. Proceedings of the 1995 Conference, The MIT Press, 1996, pp. 24–30. URL: http://papers.nips.cc/paper/1152-extracting-tree-structured-representations-of-trained-networks.pdf.

[23] G. Ciatto, R. Calegari, A. Omicini, D. Calvaresi, Towards XMAS: eXplainability through Multi-Agent Systems, in: C. Savaglio, G. Fortino, G. Ciatto, A. Omicini (Eds.), AI&IoT 2019 – Artificial Intelligence and Internet of Things 2019, volume 2502 of *CEUR Workshop Proceedings*, CEUR WS, 2019, pp. 40–53. URL: http://ceur-ws.org/Vol-2502/paper3.pdf.

[24] R. Calegari, G. Ciatto, A. Omicini, On the integration of symbolic and sub-symbolic techniques for XAI: A survey, Intelligenza Artificiale 14 (2020) 7–32. doi:10.3233/IA-190036.

[25] N. Barakat, J. Diederich, Eclectic rule-extraction from support vector machines, International Journal of Computer and Information Engineering 2 (2008) 1672–1675. doi:10.5281/zenodo.1055511.

[26] D. Martens, B. Baesens, T. Van Gestel, J. Vanthienen, Comprehensible credit scoring models using rule extraction from support vector machines, European Journal of Operational Research 183 (2007) 1466–1476. doi:10.1016/j.ejor.2006.04.051.

[27] R. Setiono, W. K. Leow, J. M. Zurada, Extraction of rules from artificial neural networks

for nonlinear regression, IEEE Transactions on Neural Networks 13 (2002) 564–577. doi:10.1109/TNN.2002.1000125.

[28] G. P. J. Schmitz, C. Aldrich, F. S. Gouws, ANN-DT: an algorithm for extraction of decision trees from artificial neural networks, IEEE Transactions on Neural Networks 10 (1999) 1392–1401. doi:10.1109/72.809084.

[29] K. Saito, R. Nakano, Extracting regression rules from neural networks, Neural Networks 15 (2002) 1279–1288. doi:10.1016/S0893-6080(02)00089-8.

[30] R. Konig, U. Johansson, L. Niklasson, G-REX: A versatile framework for evolutionary data mining, in: 2008 IEEE International Conference on Data Mining Workshops (ICDM 2008 Workshops), 2008, pp. 971–974. doi:10.1109/ICDMW.2008.117.

[31] L. Breiman, J. Friedman, C. J. Stone, R. A. Olshen, Classification and Regression Trees, CRC Press, 1984.

[32] A. A. Freitas, Comprehensible classification models: a position paper, ACM SIGKDD Explorations Newsletter 15 (2014) 1–10. doi:10.1145/2594473.2594475.

[33] J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, B. Baesens, An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models, Decision Support Systems 51 (2011) 141–154. doi:10.1016/j.dss.2010.12.003.

[34] P. M. Murphy, M. J. Pazzani, Id2-of-3: Constructive induction of m-of-n concepts for discriminators in decision trees, in: Machine Learning Proceedings 1991, Elsevier, 1991, pp. 183–187.

[35] J. R. Quinlan, C4.5: Programming for machine learning, Morgan Kauffmann (1993). URL: https://dl.acm.org/doi/10.5555/152181.

[36] J. R. Quinlan, Simplifying decision trees, International Journal of Man-Machine Studies 27 (1987) 221–234. doi:10.1016/S0020-7373(87)80053-6.

[37] R. Andrews, J. Diederich, A. B. Tickle, Survey and critique of techniques for extracting rules from trained artificial neural networks, Knowledge-Based Systems 8 (1995) 373–389. doi:/10.1016/0950-7051(96)81920-4.

[38] G. G. Towell, J. W. Shavlik, Extracting refined rules from knowledge-based neural networks, Machine Learning 13 (1993) 71–101. doi:10.1007/BF00993103.

[39] 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 (JMLR) 12 (2011) 2825–2830. URL: https://dl.acm.org/doi/10.5555/1953048.2078195.

[40] G. Ciatto, R. Calegari, E. Siboni, E. Denti, A. Omicini, 2P-Kt: logic programming with objects & functions in Kotlin, in: R. Calegari, G. Ciatto, E. Denti, A. Omicini, G. Sartor (Eds.), WOA 2020 – 21th Workshop "From Objects to Agents", volume 2706 of *CEUR Workshop Proceedings*, CEUR WS, 2020, pp. 219–236. URL: http://ceur-ws.org/Vol-2706/paper14.pdf, 21st Workshop "From Objects to Agents" (WOA 2020), Bologna, Italy, 14–16 September 2020. Proceedings.

[41] G. Ciatto, R. Calegari, A. Omicini, Lazy stream manipulation in Prolog via backtracking: The case of 2p-kt, in: W. Faber, G. Friedrich, M. Gebser, M. Morak (Eds.), Logics in Artificial Intelligence, volume 12678 of *LNCS*, Springer, 2021, pp. 407–420. doi:10.1007/978-3-030-75775-5_27, 17th European Conference, JELIA 2021, Virtual Event, May

17–20, 2021, Proceedings.

[42] J. Dougherty, R. Kohavi, M. Sahami, Supervised and unsupervised discretization of continuous features, in: A. Prieditis, S. J. Russell (Eds.), Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, USA, July 9-12, 1995, Morgan Kaufmann, 1995, pp. 194–202. doi:10.1016/b978-1-55860-377-6.50032-3.

[43] Y. Yang, G. I. Webb, X. Wu, Discretization methods, in: O. Maimon, L. Rokach (Eds.), Data Mining and Knowledge Discovery Handbook, 2nd ed, Springer, 2010, pp. 101–116. doi:10.1007/978-0-387-09823-4_6.

[44] R. Kerber, Chimerge: Discretization of numeric attributes, in: W. R. Swartout (Ed.), Proceedings of the 10th National Conference on Artificial Intelligence, San Jose, CA, USA, July 12-16, 1992, AAAI Press / The MIT Press, 1992, pp. 123–128. URL: http://www.aaai.org/Library/AAAI/1992/aaai92-019.php.

[45] K. M. Ho, P. D. Scott, Zeta: A global method for discretization of continuous variables, in: D. Heckerman, H. Mannila, D. Pregibon (Eds.), Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97), Newport Beach, California, USA, August 14-17, 1997, AAAI Press, 1997, pp. 191–194. URL: http://www.aaai.org/Library/KDD/1997/kdd97-037.php.

[46] M. Boullé, Khiops: A statistical discretization method of continuous attributes, Machine Learning 55 (2004) 53–69. doi:10.1023/B:MACH.0000019804.29836.05.

[47] L. A. Kurgan, K. J. Cios, CAIM discretization algorithm, IEEE Trans. Knowl. Data Eng. 16 (2004) 145–153. doi:10.1109/TKDE.2004.1269594.

[48] A. Cano, D. T. Nguyen, S. Ventura, K. J. Cios, ur-caim: improved CAIM discretization for unbalanced and balanced data, Soft Comput. 20 (2016) 173–188. doi:10.1007/s00500-014-1488-1.

[49] H. Elhilbawi, S. Eldawlatly, H. Mahdi, The importance of discretization methods in machine learning applications: A case study of predicting ICU mortality, in: A. E. Hassanien, K. Chang, M. Tang (Eds.), Advanced Machine Learning Technologies and Applications - Proceedings of AMLTA 2021, Cairo, Egypt, March 22-24, 2021, volume 1339 of *Advances in Intelligent Systems and Computing*, Springer, 2021, pp. 214–224. doi:10.1007/978-3-030-69717-4_23.

[50] M. Gavanelli, F. Rossi, Constraint logic programming, in: A. Dovier, E. Pontelli (Eds.), A 25-Year Perspective on Logic Programming: Achievements of the Italian Association for Logic Programming, GULP, volume 6125 of *LNCS*, Springer, 2010, pp. 64–86. doi:10.1007/978-3-642-14309-0_4.

[51] J. Jaffar, M. J. Maher, Constraint logic programming: A survey, J. Log. Program. 19/20 (1994) 503–581. doi:10.1016/0743-1066(94)90033-7.

[52] D. Calvaresi, G. Ciatto, A. Najjar, R. Aydoğan, L. Van der Torre, A. Omicini, M. Schumacher, Expectation: Personalized explainable artificial intelligence for decentralized agents with heterogeneous knowledge, in: D. Calvaresi, A. Najjar, M. Winikoff, K. Främling (Eds.), Explainable and Transparent AI and Multi-Agent Systems. Third International Workshop, EXTRAAMAS 2021, Virtual Event, May 3–7, 2021, Revised Selected Papers, volume 12688 of *LNCS*, Springer Nature, Basel, Switzerland, 2021, pp. 331–343. URL: http://link.springer.com/10.1007/978-3-030-82017-6_20. doi:10.1007/978-3-030-82017-6_20.