Untying black boxes with clustering-based symbolic knowledge extraction

Undefined 1 (2024) 1–14
IOS Press

Untying Black Boxes with Clustering-Based Symbolic Knowledge Extraction

Federico Sabbatini [a,*] and Roberta Calegari [b]
[a] *Dipartimento di Scienze Pure e Applicate (DiSPeA), Università di Urbino, Urbino, Italy*
E-mail: f.sabbatini1@campus.uniurb.it
[b] *Dipartimento di Informatica – Scienza e Ingegneria (DISI), Università di Bologna, Italy*
E-mail: roberta.calegari@unibo.it

Abstract. Machine learning black boxes, exemplified by deep neural networks, often exhibit challenges in interpretability due to their reliance on complicated relationships involving numerous internal parameters and input features. This lack of transparency from a human perspective renders their predictions untrustworthy, particularly in critical applications. In this paper, we address this issue by introducing the design and implementation of CReEPy, an algorithm for symbolic knowledge extraction based on explainable clustering. Specifically, CReEPy leverages the underlying clustering performed by the ExACT or CREAM algorithms to generate human-interpretable Prolog rules that mimic the behaviour of opaque models. Additionally, we introduce CRASH, an algorithm for the automated tuning of hyper-parameters required by CReEPy. We present experiments evaluating both the human readability and predictive performance of the proposed knowledge-extraction algorithm, employing existing state-of-the-art techniques as benchmarks for comparison in real-world applications.

Keywords: Explainable clustering, Explainable artificial intelligence, Symbolic knowledge extraction, PSyKE

1. Introduction

There has been a growing demand for transparency in recent years, particularly in critical domains [14,15]. This demand has led to a lack of trust amongst humans in predictions obtained from machine learning (ML) models that lack interpretability. Such models are often referred to as *opaque* or *black boxes* (BBs) due to their opacity. While complex ML models tend to offer superior predictive performance, they pose challenges when it comes to human inspection. Consequently, the use of opaque models for high-stakes decisions necessitates the derivation of human-intelligible knowledge to ensure accountability and understanding.

To retain the predictive capabilities of ML models, many strategies for achieving explainable behaviours have been proposed in the literature [2,19]. These include the adoption of interpretable ML predictors [27] and mechanisms designed to reverse-engineer the predictors' behaviour [23]. Symbolic knowledge-extraction (SKE) techniques play a major role in this context, operating in a post-processing phase to distill interpretable knowledge from a BB predictor. Building upon recent advancements in the SKE field [8], we present CReEPy, a novel, general-purpose knowledge-extraction algorithm based on interpretable clustering. CReEPy applies to any type of BB predictor.

CReEPy is built upon the ExACT and CREAM clustering algorithms. It pedagogically [1] explains BBs performing classification or regression tasks and operating on continuous input features. CReEPy proves the effectiveness of exploiting explainable clustering to achieve the interpretability of BBs. Indeed, it enables the extraction of more concise and accurate explanations compared to analogous state-of-the-art techniques.

Since to execute CReEPy a set of parameters is required (e.g., the chosen explainable clustering algorithm and, in turn, the corresponding hyper-parameters),

*Corresponding author. E-mail: f.sabbatini1@campus.uniurb.it.

we also designed an automated tuning algorithm, named CRASH.

Accordingly, the paper is organised as follows: Section 2 introduces background information on the topics discussed here and related works present in the literature. Sections 3 and 4 describe the CReEPy and CRASH algorithms, respectively. Experiments and benchmark comparisons are discussed in Section 5. Finally, conclusions are drawn in Section 6.

2. Related Works

2.1. Symbolic Knowledge Extraction

SKE consists of obtaining human-interpretable rules out of BB predictors using a surrogate, explainable model that is capable of mimicking the BB, named in this context *underlying model*. The underlying model may be a classifier, a regressor, a clustering technique, or any other opaque predictor. The mimicking capabilities of the surrogate model are assessed via the comparison of the outputs provided by the underlying and surrogate models with respect to the same inputs. SKE techniques are currently applied in a wide range of contexts [3,6,18,20,21,42,43,45].

The construction of the surrogate predictor may be performed in a decompositional or pedagogical way [1]. In the former case, the BB kind and internal structure are considered, so these algorithms are not general and can be applied only to a subset of BBs, e.g., RefAnn [44] accepts as underlying predictors only neural networks having a single hidden layer. On the other hand, pedagogical techniques only consider the underlying BB input/output relationship and thus they are more general and present no constraints on the BB type and complexity.

In the following, we provide a brief description of the SKE algorithms chosen as benchmarks for the experiments presented in this work.

2.1.1. ITER

ITER [22] is a pedagogical knowledge-extraction algorithm explicitly designed for black-box regressors. It extracts knowledge in the form of rule lists while imposing no constraint on the nature, structure, or training of the underlying opaque model.

To extract rules, the ITER algorithm steps through the creation and iterative expansion of several disjoint hypercubes, covering the whole input space the regressor has been trained upon. In other words, ITER accepts as input a regressor and the data set used for its training, then iteratively partitions the input feature space following a bottom-up strategy.

At the end of the process, each partition is converted into a human-interpretable rule associated with a constant output value.

2.1.2. GridEx and GridREx

The GridEx algorithm [40] is a pedagogical technique performing symbolic knowledge extraction from BBs designed for regression tasks. Thanks to the generalisation proposed in [37,39] it can be also applied to explain classifiers. In both cases, data sets have to be described by continuous features. It draws inspiration from ITER, intending to address the challenges arising from its potential slow convergence and limitations in both input space coverage and fidelity when applied to high-dimensional data sets. GridEx satisfies this goal by relying on a top-down partitioning strategy, thus achieving good results in terms of both the number of extracted rules and corresponding predictive performance with respect to the underlying BB and the data.

The partitioning strategy adopted by GridEx consists of the recursive input feature space splitting into smaller subregions according to a similarity threshold. At the end of the partitioning, each region is translated into a human-readable rule, having preconditions describing the region and a postcondition representing the associated output value, which is a constant obtained by averaging the underlying model predictions for the samples included in the region.

Unfortunately, in some real-world applications, the undesired discretisation introduced by the constant outputs of GridEx may hinder the predictive performance of the extractor. GridREx [28] overcomes this issue by training a linear model inside each identified hypercubic region. Linear models are fitted on the instances contained in the corresponding hypercubes and each cube is associated with a rule having a set of conditions on the input variables as antecedent part, equally to ITER and GridEx, and a linear combination of the input variables (given by the linear model) as consequent part. As a result, output predictions given by the extracted rules are no more averaged output values of the samples contained in the corresponding hypercubes, but more accurate linear equations. Given the nature if its predictions, GridREx only supports regression tasks.

A disadvantage shared by GridEx and GridREx is that they perform a *symmetric* partitioning – i.e., dur-

ing a given iteration, they split each input dimension in a given number of congruent partitions. Therefore, this strategy may lead to suboptimal solutions when applied to real-world data sets.

2.1.3. REAL

Rule-extraction-as-learning (REAL; [11]) is a pedagogical SKE procedure to explain BB classifiers operating upon binary input features. However, by performing an upstream feature binarisation, it is possible to adopt it also for other kinds of input attributes, e.g., discrete or even continuous. REAL aims at extracting human-interpretable lists of conjunctive rules via a learning process based on sampling and queries. Output rules are mainly *if-then* rules where the post-condition is a class label and the pre-conditions are Boolean predicates over individual input features. Pre-conditions usually concern a subset of the input features, previously generalised by dropping redundant and/or non-discriminant antecedents.

2.1.4. CART

CART [7] is not properly a SKE technique, since it is based on the induction of binary decision trees on data set instances. However, it may be applied as well to the output of a BB predictor to obtain a decision tree representing the BB behaviour. Starting from the tree, it is straightforward to extract human-comprehensible rules by converting each possible path from the tree root to the leaves into a logic rule. CART can be utilised for both black-box classifiers and regressors; nevertheless, in this scenario, the output value remains constant. Consequently, predictions may experience undesirable discretisation effects when employed in regression tasks.

2.2. Explainable Clustering via EXACT *and* CREAM

EXACT [31] is an algorithm performing explainable clustering. It merges the aggregation strategies found in traditional clustering techniques with the cluster assignment approach using decision trees, similar to other explainable or interpretable clustering procedures [4,5,12,17]. For this reason, with EXACT it is possible to obtain explainable clusters by inducing a top-down decision tree over the training data according to a strictly hierarchical strategy. Indeed, identified clusters have the peculiarity of being concentric. The strategy adopted for the tree's internal nodes is to use hypercubic splits to separate whole clusters of data while avoiding the presence of instances from multiple clusters inside the same hypercubic region.

To partition the input space, EXACT adopts Gaussian mixture models (GMMs; [26]) to find clusters and DBSCAN [13,24] to remove the outliers from these clusters, before approximating them with hypercubes. Explainability is achieved thanks to this approximation of each identified cluster. The concentric nature of the EXACT's hierarchical approximations enables the creation of a global interpretable clustering in the form of a rule list, where each cluster is simply expressed through a rule having a single hypercube inclusion constraint, starting from the innermost cluster through the outermost. The same structure may be used to provide local explanations for single clustering assignments.

CREAM [32] extends EXACT by providing a more complex splitting strategy based on the iterative greedy minimisation of the predictive error measured for each possible split.

Users may leverage the automated ORCHID procedure [32] to find optimum values for the hyper-parameters required by both EXACT and CREAM.

3. SKE via Explainable Clustering with CReEPy

In this section, we introduce the design and implementation of a novel knowledge-extraction technique, named CReEPy (Clustering-based REcursive Extraction as a PYramid; [35]), capable of deriving human-interpretable rules in Prolog syntax from BB models of any type and suitable for both classification and regression tasks. Aligned with the concept presented in [30,34], CReEPy performs knowledge extraction by employing a preliminary interpretable clustering technique (i.e., EXACT or CREAM) on the training data. To extract knowledge from a predictive model, the data set output feature (i.e., the ground truth) is replaced with the opaque predictions provided by the BB. CReEPy is also suitable for performing rule induction when directly applied to a data set. In the first case, CReEPy explains the predictions through human-interpretable knowledge. Otherwise, it provides interpretable relationships between the data set's input and output attributes.

3.1. The CReEPy Algorithm

CReEPy has been designed to be independent of the underlying clustering method. Consequently, it can be

Algorithm 1 CReEPy pseudocode

Require: predictor P
Require: data set D
Require: underlying clustering technique Ξ
Require: underlying clustering parameters Π
Require: input feature relevance set Φ
Require: input feature relevance threshold Θ

function CReEPy(P, D, Ξ, Π, Φ, Θ)
  $D' \leftarrow$ CreateDataset(P, D)
  $regions \leftarrow$ Clustering(Ξ, Π, D')
  return $\bigcup_{r \in regions}$ { RegionToRule(r, Φ, Θ) }

function Clustering(Ξ, Π, D)
  $instance \leftarrow$ Init(Ξ, Π)
  $instance \leftarrow$ Train($instance$, D)
  return Clusters($instance$)

function Init(Ξ, Π)
  return an instance of Ξ parametrised with Π

function Train($clustering$, D)
  return $clustering$ trained upon D

function Clusters($clustering$)
  return the clusters identified by $clustering$

function CreateDataset(P, D)
  return D with output feature predicted by P

function RegionToRule($region$, Φ, Θ)
  for all $dim \in region$ **do**
    if Relevance(dim, Φ) $< \Theta$ **then**
      $region \leftarrow$ Drop(dim, $region$)
  return ToProlog($region$)

function Relevance(dim, Φ)
  return relevance of dim according to Φ

function Drop($dimension$, $region$)
  drop $dimension$ from the inputs of $region$
  return $region$

function ToProlog($region$)
  $head \leftarrow \bigcup_{dim \in region}$ {name of dim}
  $body \leftarrow \bigcup_{dim \in region}$ {boundaries of dim}
  if current task is classification **then**
    $head \leftarrow head \cup$ {output label of $region$}
  else if current task is regression with constant output **then**
    $head \leftarrow head \cup$ {output value of $region$}
  else if current task is regression with linear output **then**
    $head \leftarrow head \cup$ {output variable of $region$}
    $body \leftarrow body \cup$ {equation describing $region$}
  return a Prolog rule described by $head$ and $body$

employed in conjunction with various clustering techniques, as long as they offer hypercubic approximations of the input space. Moreover, there is potential for future extensions of CReEPy to support other tree-based clustering approaches, as such methods essentially segment the input feature space with cuts perpendicular to the axes, and each path from the tree root to a leaf can be translated into a hypercube.

Being explicitly designed to work seamlessly with ExACT and CREAM, CReEPy generates logic knowledge in the form of a Prolog theory (examples of which are detailed in the experiment section). The Prolog theory effectively mimics the decision-making process of the underlying BB model, with each clause corresponding to an approximated cluster identified by ExACT or CREAM. The interpretability of the BB is enhanced through the transition from completely opaque outcomes to the generation of classification or regression rules that are both human- and machine-interpretable, describing the rationale behind the predictions.

As ExACT and CREAM clusters are structured as hierarchical cubes and difference cubes – defined through interval inclusions and exclusions – Prolog theories are particularly suitable, benefitting from the inherent ordering of clauses. Consequently, each clause can be associated only with preconditions referring to the *inclusion* in a hypercube, assuming the *exclusion* from all the cubes described by the preceding clauses as true. The expressiveness of this semantics is thus exploited at its limit by ordering the Prolog rules starting from the one associated with the innermost hypercubic region and then following the hierarchy up to the outermost region—equivalent to the surrounding cube of the data set at hand. The last rule may be extended to a default rule to achieve 100% completeness [29].

3.2. User-Defined Parameters

The human-readability extent of the output theory provided by CReEPy critically depends on the number of clauses composing the theory and, in turn, on the number of preconditions appearing in each clause. The number of clauses can be controlled by users via a hyper-parameter tuning phase, considering that these are produced based on the clusters identified with the underlying clustering techniques. As a consequence, pivotal hyper-parameters that must be set by users to obtain high-quality knowledge with CReEPy, possibly

with the aid of CRASH, are those required by EXACT and CREAM, namely:

- θ is a predictive error threshold calibrating the trade-off between predictive performance and human-readability extent (number of clauses) of the explainable clustering. Only nodes associated with predictive errors larger than the user-defined threshold are further partitioned in the successive iterations of the algorithm. This parameter should be set according to the task at hand, e.g., it may represent an upper-bound for the rate of incorrect predictions in classification tasks as well as for the mean squared error in regression tasks;
- δ is the maximum allowed depth. Given the recursive nature of EXACT and CREAM, users can tune this parameter to stop the algorithm's input space partitioning after the desired quantity of iterations. The maximum depth is not reached if the tree expansion pre-emptively terminates due to the absence of further nodes having predictive error greater than θ;
- ξ is an upper-bound for the number of clusters identifiable via GMMs during the execution of EXACT and CREAM.

It is worthwhile to point out that the clustering algorithm adopted in CReEPy is itself a hyper-parameter that may be tuned with CRASH.

The set of CReEPy's hyper-parameters is completed by an optional *input feature relevance set* and a corresponding threshold, aimed at limiting the rule preconditions to the only features with relevance greater than the threshold. Indeed, CReEPy assigns a precondition to each input dimension, i.e., an interval inclusion constraint for each input feature. Therefore, in the default version of the algorithm, each Prolog clause has n preconditions for n-dimensional data sets. This may be limiting in terms of human readability when dealing with high-dimensional data sets.

We highlight here that the input feature relevance is calculated outside CReEPy, so users are not bound to a specific method, as far as they provide the feature relevance set normalised in the [0, 1] interval. A relevance score for each input feature is mandatorily required. A suitable and fast solution to obtain these scores can be found within the Python Scikit-Learn library.[1] It is worthwhile to point out that the feature relevance threshold does not affect the underlying clustering, but only the translation into Prolog rules performed by CReEPy starting from the tree provided by EXACT or CREAM (cf. REGIONTORULE procedure in Algorithm 1).

The translation into Prolog rules is executed according to the following criteria:

- for each leaf of the tree identified via the underlying clustering technique a rule is created;
- individual rules are *if-then* logic rules where the conditional part is a conjunction of interval inclusion constraints on the input features and the corresponding action is a constant value (e.g., a class label or a number) or a linear combination of the input variables;
- constraints are defined in the internal nodes of the tree;
- actions are described in the leaves of the tree;
- all variables having relevance smaller than the user-defined threshold are removed from the conditional part of the logic rules;
- the resulting rules are converted into a theory having Prolog format, both human- and agent-interpretable.

From a predictive perspective, the quality of rules provided by CReEPy can be assessed via standard scores generally adopted for ML classification and regression tasks, e.g., accuracy and F_1 score for the former and mean absolute/squared error and R^2 score for the latter. Dedicated scoring metrics for symbolic knowledge evaluation, as Q_s and FiRe, may be used as well [33,36], to account simultaneously for fidelity and readability.

4. Automated Hyper-Parameter Tuning for CReEPy: The CRASH Optimiser

In this section we provide the details about the optimiser algorithm designed to automatise the hyper-parameter tuning of CReEPy, named CRASH (Clustering-based Rule extraction Automated Selection of Hyper-parameters) and whose workflow is resumed in Algorithm 2.

4.1. *The* CRASH *Algorithm*

CRASH is based on the iterative exploration of the hyper-parameter space to highlight the values corresponding to the *best* CReEPy instances in terms of both predictive performance and human readability.

[1] cf. https://scikit-learn.org/stable/modules/classes.html#module-sklearn.feature_selection

Algorithm 2 CRASH pseudocode

Require: predictor P
Require: data set D
Require: maximum depth Δ, default = 10
Require: max. number of Gaussian components Γ, default = 10
Require: predictive/readability loss trade-off Ψ, default 0.1
Require: maximum predictive loss increase p_{max}, default = 1.2
Require: minimum rule loss decrease r_{min}, default = 0.9
Require: patience value pat_0, default = 5

function CRASH(P, D, Δ, Γ, Ψ, $p_{max}, r_{min}, patience_0$)
  $\Pi \leftarrow \emptyset$ ▷ set of all configurations
  $O \leftarrow (\Delta, \Psi, p_{max}, r_{min}, pat_0)$ ▷ ORCHID param.
  for all $algorithm \in \{\text{EXACT}, \text{CREAM}\}$ **do**
    $\Pi \leftarrow \Pi \cup$ SEARCHALGORITHM($algorithm, D, \Gamma, \Psi, O$)

function SEARCHALGORITHM($algorithm$, D, Γ, Ψ, O)
  $\Pi \leftarrow \emptyset$ ▷ set of all configurations
  $\pi^* \leftarrow$ **undefined** ▷ best configuration
  $components \leftarrow 2$ ▷ current number of components
  while $components \leq \Gamma$ **do**
    $data \leftarrow$ SPLIT($D, components$)
    $\Pi' \leftarrow$ SEARCHCOMPONENT($D, algorithm, components, O$) ▷ current configurations
    $\pi'^* \leftarrow$ SELECTBEST(Π', Ψ) ▷ best current config.
    if SCORE(π^*) $\leq$ SCORE(π'^*) **then return** Π
    $\pi^* \leftarrow \pi'^*$
    $\Pi \leftarrow \Pi \cup \Pi'$
  return Π

function SPLIT(D, $components$)
  $n \leftarrow components \cdot 100$
  if $|D| \leq n$ **then return** D
  return n distinct random instances of D

function SEARCHCOMPONENT(D, $algorithm$, $comp$, O)
  $orchid \leftarrow$ ORCHID($D, algorithm, comp, O$)
  return $orchid.configurations$

function ORCHID(D, $algorithm$, c, O)
  return an instance of ORCHID (with O hyper-parameters) to optimise the depth and threshold parameters of $algorithm$ adopting c Gaussian components

function SELECTBEST(Π, Ψ)
  return the best configuration $\pi \in \Pi$, considering Ψ

function SCORE(π)
  return a score associated with the configuration π

The notion of best instance is defined according to the following equation:

$$\pi^* = \arg\min_{\pi \in \Pi} \{error(\pi) \cdot \lceil rules(\pi) \cdot \Psi \rceil\}, \quad (1)$$

where Π is the set of all configurations of CReEPy's parameters, π is a generic configuration, $error(\pi)$ and $rules(\pi)$ are the predictive error and the number of extracted rules, respectively, measured for the configuration π, and π^* denotes the best configuration of parameters, associated with the best CReEPy instance. We emphasise here that Equation (1) is subject to the readability/fidelity trade-off [10], which is controlled through Ψ. To elaborate, large (small) rule sets tend to have good (poor) predictive performance, resulting in small (large) predictive errors. Ψ represents the extent to which readability (expressed as the number of rules) predominates over predictive performance (expressed as predictive error) in assessing the goodness of a parameter set. A good score can be achieved by minimising both the quantity of extracted rules and the predictive error simultaneously.

In CRASH, each parameter is searched individually, e.g., by fixing all the others and studying how the predictive error and readability of CReEPy change by altering the values of that parameter. Some parameters are searched exhaustively, others with a grid search or iteratively according to ad-hoc criteria. In more detail, the optimisation is exhaustive for the underlying clustering algorithm (i.e., currently both EXACT and CREAM are tested). Numeric parameters are considered between their minimum values and corresponding user-defined upper-bounds, iteratively increased by fixed or variable quantities, possibly until a patience expiration.

CRASH may be summarised as follows:

1. pick a clustering algorithm suitable for CReEPy;
2. identify the clustering hyper-parameters tunable via automated procedures and those that cannot be automatically estimated;
3. fix all parameters of the second group to their minimum accepted values;
4. run the proper optimisation algorithm to find the best values for the remaining parameters;
5. store the best parameter configuration amongst those obtained during step 4;
6. increase the value of a parameter amongst those fixed in step 3 and repeat from step 4 until it is possible to find better configurations, until the maximum parameter values are reached, or until the patience expires;

7. repeat from step 1 with a different algorithm;
8. return the best configuration according to Equation (1).

Currently, automatically tunable clustering parameters are the maximum depth δ and the error threshold θ (step 2), which can be estimated via ORCHID (step 4). Steps 3 and 6 concern the maximum amount of Gaussian components γ. Algorithm 2 reflects the current status of CRASH, implemented according to the workflow mentioned above.

Since the execution time of CReEPy critically depends on the number of training instances (cf. Section 5.2), slicing of the training data set is performed during step 4. In particular, we fixed an upper-bound to the number of training instances equal to the number of Gaussian components $\times$ 100. This is a reasonable choice, given that n Gaussian components imply the identification of n clusters. By assuming balanced clusters, 100 training instances per cluster allow CReEPy to learn the clusters' peculiarities while remaining fast.

4.2. User-Defined Parameters

As mentioned above, to run CRASH users should define a set of parameters. We point out that the default values of these parameters are usually suitable to obtain satisfying results, comparable with or better than a manual tuning of CReEPy. The set of parameters required by CRASH depends on the clustering techniques to be tested. Currently, only the aforementioned EXACT and CREAM algorithms are supported by CReEPy. Therefore, both are examined within CRASH, and its parameters are those needed to parametrise EXACT and CREAM, tuned via the automated ORCHID procedure. When other clustering techniques are supported, we plan to add a hyperparameter to specify the set of clustering algorithms to test with CRASH, along with any other parameters needed by the new clustering methods. So far, the following optional parameters are required by CRASH:

Δ is the upper-bound for the estimation of the optimum clustering depth. The search may be pre-emptively stopped if growing depths result in no better parameter configurations. This parameter is used within ORCHID;

Γ is the upper-bound for the estimation of the optimum number of Gaussian components to be used within the explainable clustering. The search may be pre-emptively stopped if growing numbers of components result in no better parameter configurations;

Ψ is the fidelity/readability trade-off, defining the importance of the CReEPy's readability extent against that of its predictive performance. Small Ψ values tend to neglect the readability impact. For instance, if $\Psi = 0.1$ it is not relevant if a CReEPy instance produces 2 or 8 rules. Conversely, if $\Psi = 1$ the instance producing 8 rules is penalised;

pat_0 is the patience to adopt in the absence of better parameter configurations when iteratively estimating the optimum predictive error threshold of EXACT and CREAM. This parameter is used within ORCHID;

p_{max}, r_{min} are the maximum predictive error increase and the minimum readability enhancement, respectively, that are accepted/required during the estimation of the clustering error threshold. These parameters are used within ORCHID.

5. Experiments

Experiments to assess the capabilities of CReEPy applied to classification and regression tasks in comparison with state-of-the-art analogous techniques are described in the following. All the adopted implementations are included within the PSyKE framework[2] [9,38,41].

5.1. Predictive Performance and Readability Assessments

To assess the capabilities of CReEPy in explaining opaque ML predictors we carried out several experiments involving real-world data sets. We selected the Iris data set[3] [16] as a case study for classification and 6 data sets from real use cases taken from the StairwAI EU Project[4] to test CReEPy in regression tasks.

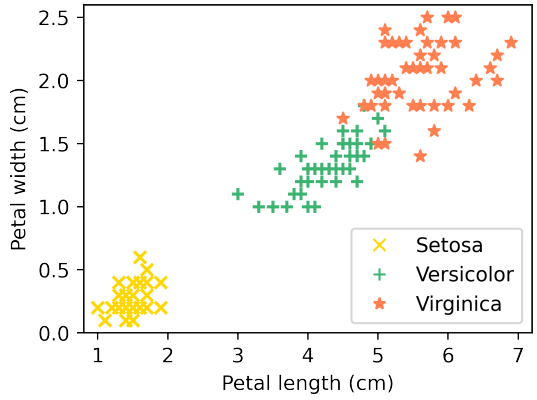
5.1.1. Classification: The Iris Data Set Case Study

The Iris data set represents a simple classification task with 4 continuous input features expressing as

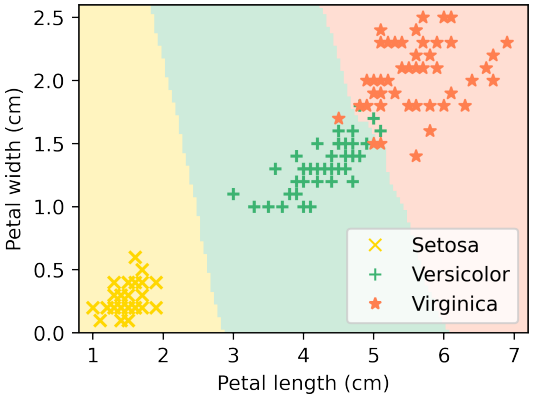
[2]Code available at `https://github.com/psykei/psyke-python`

[3]`https://archive.ics.uci.edu/ml/datasets/iris`

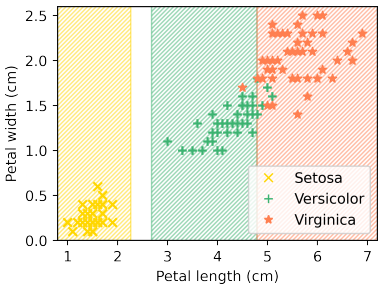
[4]`https://cordis.europa.eu/project/id/101017142`; data sets are publicly available at `https://zenodo.org/record/5838437`



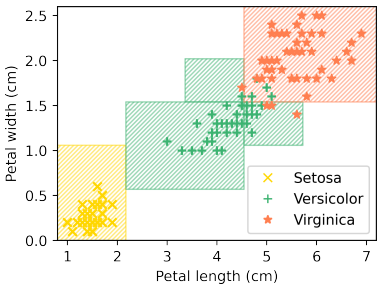
(a) Iris data set.



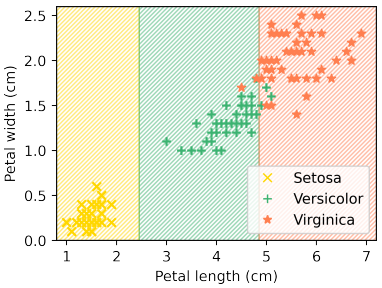
(b) 7-NN.



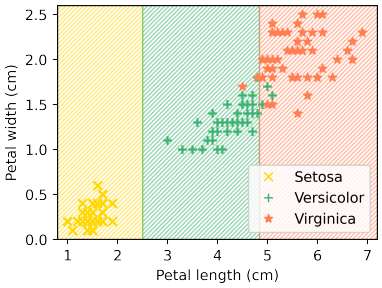
(c) GridEx, 3 rules.



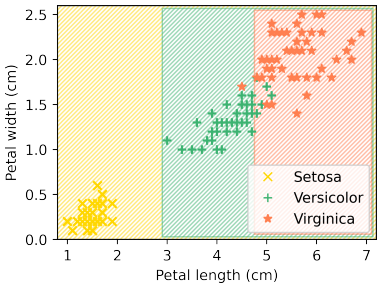
(d) GridEx, 5 rules.



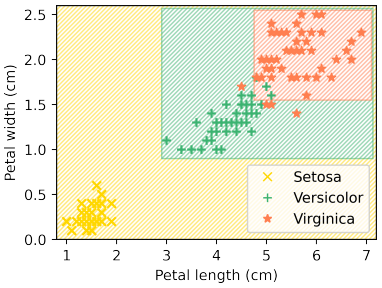
(e) CART.



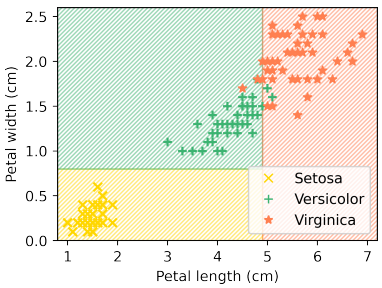
(f) ITER.



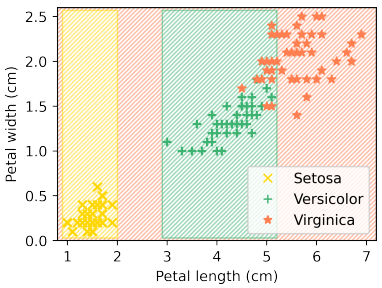
(g) CReEPy + EXACT, feature relevance threshold = 0.99.



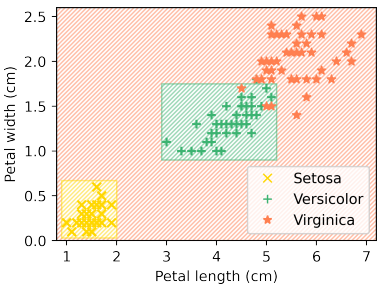
(h) CReEPy + EXACT, feature relevance threshold = 0.80.



(i) REAL.



(j) CReEPy + CREAM, feature relevance threshold = 0.99.



(k) CReEPy + CREAM, feature relevance threshold = 0.80.

Figure 1. Symbolic knowledge extraction performed on the Iris data set.

Listing 1 Rules extracted with CReEPy using ExACT for the Iris data set. Feature relevance threshold = 0.99.

```
iris(PetalLength, PetalWidth, SepalLength, SepalWidth, virginica) :-
     PetalLength in [4.75, 6.90].
iris(PetalLength, PetalWidth, SepalLength, SepalWidth, versicolor) :-
     PetalLength in [2.90, 6.90].
iris(PetalLength, PetalWidth, SepalLength, SepalWidth, setosa) :-
     PetalLength in [1.10, 6.90].
```

Listing 2 Rules extracted with CReEPy using ExACT for the Iris data set. Feature relevance threshold = 0.80.

```
iris(PetalLength, PetalWidth, SepalLength, SepalWidth, virginica) :-
     PetalLength in [4.75, 6.90], PetalWidth in [1.55, 2.60].
iris(PetalLength, PetalWidth, SepalLength, SepalWidth, versicolor) :-
     PetalLength in [2.90, 6.90], PetalWidth in [0.90, 2.60].
iris(PetalLength, PetalWidth, SepalLength, SepalWidth, setosa) :-
     PetalLength in [1.10, 6.90], PetalWidth in [0.00, 2.60].
```

many characteristics of iris flowers. The target is the species of the flowers, which in this specific context may assume 3 possible distinct values. The data set is reported in Figure 1a. In all panels of Figure 1 only the 2 most relevant input features are shown, i.e., petal length and width expressed in cm.

Our experiments on this data set are based on a k-nearest neighbours (k-NN) opaque classifier, having $k = 7$. The corresponding decision boundaries are depicted in Figure 1b.

Decision boundaries identified by CReEPy and other SKE techniques are reported in the other panels of Figure 1. The hyper-parameters used for each SKE technique are listed in the following.

GridEx We adopted 2 different instances of GridEx. The one corresponding to Figure 1c produces 3 output rules (one per possible output class) and performs 14 slices only along input features having importance greater than 0.99—i.e., only along the most important feature, the petal length. Conversely, the GridEx instance shown in Figure 1d performs 5 slices along each input dimension having importance greater than 0.80—i.e., petal length and width. This results in the 5 output rules depicted in the figure.

CART The decision boundaries shown in Figure 1e are obtained by growing an unbounded CART decision tree (no constraints on the tree depth, nor on the leaf quantity).

ITER The ITER instance producing the input space partitioning depicted in Figure 1f has been tuned with a minimum cube update of 0.2 and an error threshold of 0.1. The maximum number of allowed iterations and the minimum quantity of samples to consider in each cube have been fixed to 600 and 150, respectively. The algorithm started from a single random cube.

REAL The input space partitioning obtained with REAL is depicted in Figure 1i. No parameter tuning is required for this algorithm.

CReEPy Figures 1g and 1h correspond to CReEPy instances with feature relevance thresholds equal to 0.99 and 0.80, respectively, and exploiting ExACT as underlying clustering technique. The former implies considering only the most relevant feature when performing the knowledge extraction. By relaxing the threshold to 0.80 the second most important input feature is considered, as for GridEx. Analogously, Figures 1j and 1k show CReEPy instances adopting the CREAM underlying clustering with different parametrisations of the input feature importance threshold. The input space partitioning reported in Figure 1g is equivalent to the Prolog theory shown in Listing 1. The Prolog theory corresponding to the decision boundaries of Figure 1h is shown in Listing 2. The values of the other parameters required by CReEPy have been estimated with CRASH. As a result, all CReEPy instances rely on underlying clusterings using 2 Gaussian components, an error threshold equal to 0.02 and a maximum depth of 2.

Table 1 summarises the predictive performance measured for each SKE technique. The number of extracted rules is also listed as an index of the human-interpretability extent. Furthermore, we adopted the Jaccard index [25] to compare the decision boundaries of the black-box to those of the knowledge-extraction techniques. Essentially, the Jaccard index is a measure of intersection over union, calculated as follows for

each of the Iris classes (S = Setosa, Ve = Versicolor, Vi = Virginica):

$$Jaccard_c = \frac{| \mathcal{D}_c^p \cap \mathcal{D}_c^e |}{| \mathcal{D}_c^p \cup \mathcal{D}_c^e |}, \quad (2)$$

where c is the Iris class and $\mathcal{D}_c^p$, $\mathcal{D}_c^e$ are the sets of Iris instances predicted as c by the BB and the extractor, respectively. It is worth noting that, currently, there are no quantitative metrics in the literature specifically designed to assess symbolic knowledge similarity. We chose the Jaccard index for this purpose because it focuses on subregions of the input feature space characterised by the existence of data set instances while ignoring other subregions characterised by implausible input feature values.

From the results shown in Table 1, it is evident that CReEPy, compared to other state-of-the-art analogous techniques, can achieve comparable or even better predictive performance. As for the number of extracted rules, CReEPy provides 3 rules, the optimum result given that it is applied to a classification task having 3 possible outcomes. The similarity exhibited by the decision boundaries obtained with CReEPy with respect to the BB aligns with that observed for the other state-of-the-art competitors (> 90%).

5.1.2. Regression: The StairwAI Case Study

Thanks to the versatility of the underlying clustering constituting the core of CReEPy, it is possible to apply this latter to regression tasks as well. All the data sets used here as case studies are composed of continuous features; 2 of them have 5 input features, and the remaining have 1 input feature. A different BB has been applied to each data set to draw predictions. A comparison between the knowledge extraction performed on the aforementioned data sets by CReEPy and other state-of-the-art analogous methods (namely, GridEx, GridREx and CART) has been reported in Table 3. Each measurement has been averaged on 5 different executions run under analogous conditions. Results provided by different executions are almost identical or very close, so we omitted the results' standard deviation in the table. For each data set, the number of input variables and the mean absolute error (MAE) of the corresponding BB model are listed. For each extractor, the number of output rules (R) and the mean absolute error with respect to both the actual data (D) and the BB predictions are shown. For all these experiments we chose EXACT as the underlying clustering technique and local linear combinations of the input variables as outputs for the regions approximated by the EXACT instances. Indeed, the adoption of constant outputs resulted in more concise output rules having, however, far worse predictive performance.

It is important to note that only the MAE is reported in Table 3 as a measure of predictive performance, since other metrics such as the mean squared error or the R^2 score were entirely aligned with the MAE and expressed the same quality ranking. The number of extracted rules is taken as a readability measure since readability for humans decreases if the quantity of rules increases. Another index used to assess and compare the quality of extractors is the completeness of the extracted knowledge [36], but in this particular case study is not relevant, since all the procedures achieve a level of completeness above 99%.

CReEPy proved to be superior to CART from a predictive performance perspective since local linear combinations of input variables better approximate the data set/BB outputs than constant values. Furthermore, a readability comparison between the two extractors shows that CReEPy is able to halve the extracted rule quantity in 50% of experiments. Analogous considerations hold for the comparison with GridEx, with even a more evident readability enhancement when considering CReEPy.

The most interesting comparison is with GridREx, able as well to provide local approximations in the form of linear input variable combinations. By exploiting CReEPy it is possible to achieve approximately the same predictive performance shown by GridREx with far better readability (for instance, 2 output rules instead of 5 or 4, by considering experiments on data sets #2, #5 and #6). In conclusion, our proposed knowledge extractor performing an upstream interpretable clustering via EXACT is absolutely competitive with state-of-the-art SKE algorithms.

5.2. Computational Time Assessments

Our experiments are completed by a quantitative assessment of the computational time required by CReEPy to perform the knowledge extraction. Tests consider regression data set #1, by performing both row and column slicing on it. In particular, a comparison of the computational time required to handle the data set with different numbers of input features and instances has been performed. Results are reported in Figure 2. Measurements have been averaged upon 100 executions.

Table 1
Assessments for the SKE techniques applied to a 7-NN performing classification on the Iris data set.

Model	Extracted rules	Predictive performance		Similarity (Jaccard score)			
		F_1 score (data)	F_1 score (BB)	S	Ve	Vi	All
7-NN	–	0.95					
GridEx	3	0.97	0.94	1.00	0.89	0.89	0.92
GridEx	5	0.94	0.92	1.00	0.88	0.87	0.91
CART	3	0.94	0.97	1.00	0.92	0.92	0.95
ITER	3	0.94	0.97	1.00	0.92	0.92	0.95
REAL	3	0.94	0.97	1.00	0.92	0.92	0.95
CReEPy + EXACT, feature relevance threshold = 0.99	3	0.95	0.97	1.00	0.88	0.89	0.92
CReEPy + EXACT, feature relevance threshold = 0.80	3	0.94	0.96	1.00	0.88	0.89	0.92
CReEPy + CREAM, feature relevance threshold = 0.99	3	0.90	0.94	1.00	0.88	0.86	0.91
CReEPy + CREAM, feature relevance threshold = 0.80	3	0.97	0.96	1.00	0.88	0.86	0.91

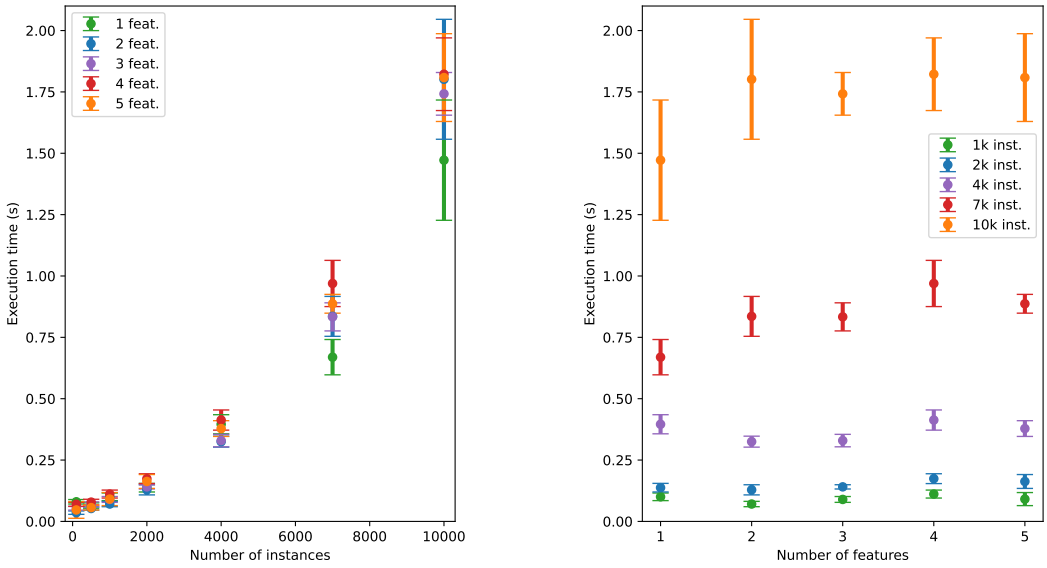
Table 2
Regression data sets used to test CReEPy and compare it to analogous techniques. For each data set are reported: a unique identifier, the name of the data set, the number of considered input features, the name of the considered output feature, and the mean absolute error measured for the BB trained on the data set.

ID	Name	Input variables	Output variables	BB MAE
#1	Anticipate	5	cost	0.4
#2	Anticipate	1	memory	4.1
#3	Anticipate	1	time	8.3
#4	Contingency	5	cost	1.5
#5	Contingency	1	memory	3.6
#6	Contingency	1	time	0.8

Table 3
Results of CReEPy applied to the 6 data sets described in Table 2. For each data set the number of extracted rules (R) and the MAE with respect to the data (D) and the underlying BB model are provided. Results are compared with those of GridREx, GridEx, and CART applied to the same data sets. Best results are highlighted in bold.

Data set	CReEPy			GridREx			GridEx			CART		
		MAE			MAE			MAE			MAE	
	R	D	BB	R	D	BB	R	D	BB	R	D	BB
#1	**3**	**1.5**	**1.5**	5	1.9	2.0	5	14.6	14.6	4	14.7	14.7
#2	**2**	**4.9**	3.3	5	**4.9**	**3.2**	5	15.0	14.7	4	17.4	17.0
#3	**3**	11.5	7.9	4	**11.1**	**6.6**	5	17.7	15.0	4	16.7	12.9
#4	**4**	26.6	26.8	**4**	**24.4**	**24.6**	5	28.5	28.6	**4**	25.1	25.1
#5	**2**	4.7	**2.3**	4	**4.5**	**2.3**	4	4.7	**2.3**	4	**4.5**	**2.3**
#6	**2**	**1.0**	**0.7**	5	**1.0**	**0.7**	5	3.1	3.1	4	3.9	3.8

From Figure 2a it is clear that the execution time grows by augmenting the number of training instances. Clues on the independence of required time with respect to the number of input features may be found in the same figure. Such independence is clearly noticeable in Figure 2b, showing that the computational time is always smaller than 2, 1 and 0.5 seconds for 10 000, 7 000 and 4 000 instances, respectively, regardless of the input feature quantity. In conclusion, we suggest fastening CReEPy, when necessary, by reducing the



(a) Feature number as parameter.

(b) Instance number as parameter.

Figure 2. Execution time of CReEPy with respect to the number of input features of the domain and the number of instances adopted for the training.

number of training data points instead of the number of input features.

6. Conclusions

In this paper, we introduce CReEPy for SKE, applicable to any kind of opaque ML classifier or regressor operating on data sets characterised by continuous input features. CReEPy demonstrates superior performance compared to existing techniques in terms of predictive accuracy and human interpretability. The algorithm operates in two phases: it applies an explainable clustering technique to the training data, followed by the knowledge extraction phase. The human readability of the extracted knowledge is ensured by presenting it to users in the form of a logic theory adhering to the Prolog syntax.

We employ two upstream clustering techniques, namely ExACT and CREAM, specifically designed for CReEPy. These algorithms exploit GMMs and DBSCAN to identify clusters and approximate them with human-interpretable hypercubic regions described by interval inclusion constraints on the input features.

Furthermore, we introduce CRASH, a dedicated procedure aimed at automatically tuning the hyperparameters required by CReEPy, along with the underlying clustering technique. This ensures that CReEPy produces high-quality knowledge.

Our future research endeavours will concentrate on developing more sophisticated and effective clustering techniques compatible with CReEPy. Specifically, we aim to enhance the rationale behind region approximation in ExACT and CREAM, as well as address the limitations associated with the application of GMMs and DBSCAN in the current versions of the algorithms. We also plan to exploit fuzzy rules to modify the format of the human-interpretable rules presented to end-users and to separate the outputs provided by CReEPy from the ordered rule lists currently used, especially to support local explanations. Our final aim is to conceive an efficient method to translate the ordered rule lists provided by CReEPy into unordered sets, without noticeable conciseness losses. These efforts will enhance the knowledge interpretability extent for all users, regardless of their technical background.

Acknowledgments

This work has been supported by the EU ICT-48 2020 project TAILOR (No. 952215) and the European Union's Horizon Europe AEQUITAS research and innovation programme under grant number 101070363.

Conflict of Interests

The authors have no conflict of interest to declare.

References

[1] Robert Andrews, Joachim Diederich, and Alan B. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8(6):373–389, 1995.

[2] Stéphane Ayache, Rémi Eyraud, and Noé Goudian. Explaining black boxes on sequential data using weighted automata. In *International Conference on Grammatical Inference*, pages 81–103. PMLR, 2019.

[3] Bart Baesens, Rudy Setiono, Christophe Mues, and Jan Vanthienen. Using neural network rule extraction and decision tables for credit-risk evaluation. *Management Science*, 49(3):312–329, 2003.

[4] Jayanta Basak and Raghu Krishnapuram. Interpretable hierarchical clustering by constructing an unsupervised decision tree. *IEEE Trans. Knowl. Data Eng.*, 17(1):121–132, 2005.

[5] Dimitris Bertsimas, Agni Orfanoudaki, and Holly M. Wiberg. Interpretable clustering via optimal trees. *CoRR*, abs/1812.00539, 2018.

[6] Guido Bologna and Christian Pellegrini. Three medical examples in neural network rule extraction. *Physica Medica*, 13:183–187, 1997.

[7] Leo Breiman, Jerome Friedman, Charles J. Stone, and Richard A. Olshen. *Classification and Regression Trees*. CRC Press, 1984.

[8] Roberta Calegari, Giovanni Ciatto, and Andrea Omicini. On the integration of symbolic and sub-symbolic techniques for XAI: A survey. *Intelligenza Artificiale*, 14(1):7–32, 2020.

[9] Roberta Calegari and Federico Sabbatini. The PSyKE technology for trustworthy artificial intelligence. 13796:3–16, March 2023. XXI International Conference of the Italian Association for Artificial Intelligence, AIxIA 2022, Udine, Italy, November 28 – December 2, 2022, Proceedings.

[10] Giovanni Ciatto, Federico Sabbatini, Andrea Agiollo, Matteo Magnini, and Andrea Omicini. Symbolic knowledge extraction and injection with sub-symbolic predictors: A systematic literature review. *ACM Computing Surveys*, 56(6):161:1–161:35, 2024.

[11] Mark W. Craven and Jude W. Shavlik. Using sampling and queries to extract rules from trained neural networks. In *Machine Learning Proceedings 1994*, pages 37–45. Elsevier, 1994.

[12] Sanjoy Dasgupta, Nave Frost, Michal Moshkovitz, and Cyrus Rashtchian. Explainable k-means and k-medians clustering. *CoRR*, abs/2002.12538, 2020.

[13] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Evangelos Simoudis, Jiawei Han, and Usama M. Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, pages 226–231. AAAI Press, 1996.

[14] European Commission. AI Act – Proposal for a regulation of the european parliament and the council laying down harmonised rules on artificial intelligence (Artificial Intelligence Act) and amending certain union legislative acts. `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52021PC0206`, 2021.

[15] European Commission, Content Directorate-General for Communications Networks, and Technology. *Ethics guidelines for trustworthy AI*. Publications Office, 2019.

[16] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.

[17] Ricardo Fraiman, Badih Ghattas, and Marcela Svarc. Interpretable clustering using unsupervised binary trees. *Adv. Data Anal. Classif.*, 7(2):125–145, 2013.

[18] Leonardo Franco, José Luis Subirats, Ignacio Molina, Emilio Alba, and José 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*, pages 1004–1011. Springer, 2007.

[19] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Computing Surveys*, 51(5):1–42, 2018.

[20] Yoichi Hayashi, Rudy Setiono, and Katsumi Yoshida. A comparison between two neural network rule extraction techniques for the diagnosis of hepatobiliary disorders. *Artificial intelligence in Medicine*, 20(3):205–216, 2000.

[21] Alexander Hofmann, Carsten Schmitz, and Bernhard Sick. Rule extraction from neural networks for intrusion detection in computer networks. In *2003 IEEE International Conference on Systems, Man and Cybernetics*, volume 2, pages 1259–1265. IEEE, 2003.

[22] Johan Huysmans, Bart Baesens, and Jan Vanthienen. ITER: An algorithm for predictive regression rule extraction. In *Data Warehousing and Knowledge Discovery (DaWaK 2006)*, pages 270–279. Springer, 2006.

[23] Eoin M. Kenny, Courtney Ford, Molly Quinn, and Mark 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*, 294:103459, 2021.

[24] Robert F. Ling. On the theory and construction of k-clusters. *The Computer Journal*, 15(4):326–332, 01 1972.

[25] Allan H. Murphy. The Finley affair: A signal event in the history of forecast verification. *Weather and forecasting*, 11(1):3–20, 1996.

[26] Kevin P. Murphy. *Machine learning – A probabilistic perspective*. Adaptive computation and machine learning series. MIT Press, 2012.

[27] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

[28] Federico Sabbatini and Roberta Calegari. Symbolic knowledge extraction from opaque machine learning predictors: GridREx & PEDRO. In Gabriele Kern-Isberner, Gerhard Lakemeyer, and Thomas Meyer, editors, *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning, KR 2022, Haifa, Israel. July 31 - August 5, 2022*, 2022.

[29] Federico Sabbatini and Roberta Calegari. Achieving complete coverage with hypercube-based symbolic knowledge-extraction techniques. In Slawomir Nowaczyk, Przemyslaw

Biecek, Neo Christopher Chung, Mauro Vallati, Pawel Skruch, Joanna Jaworek-Korjakowska, Simon Parkinson, Alexandros Nikitas, Martin Atzmüller, Tomás Kliegr, et al., editors, *Artificial Intelligence. ECAI 2023 International Workshops – XAI^3, TACTIFUL, XI-ML, SEDAMI, RAAIT, AI4S, HYDRA, AI4AI, Kraków, Poland, September 30 – October 4, 2023, Proceedings, Part I*, volume 1947 of *Communications in Computer and Information Science*, pages 179–197. Springer, 2023.

[30] Federico Sabbatini and Roberta Calegari. Bottom-up and top-down workflows for hypercube- and clustering-based knowledge extractors. In Davide Calvaresi, Amro Najjar, Andrea Omicini, Reyhan Aydogan, Rachele Carli, Giovanni Ciatto, and Kary Främling, editors, *Explainable and Transparent AI and Multi-Agent Systems. Fifth International Workshop, EXTRAAMAS 2023, London, UK, May 29, 2023, Revised Selected Papers*, volume 14127 of *LNCS*, pages 116–129, Basel, Switzerland, 2023. Springer Cham.

[31] Federico Sabbatini and Roberta Calegari. ExACT explainable clustering: Unravelling the intricacies of cluster formation. In Clayton K. Baker, Lucía Gómez Álvarez, Jesse Heyninck, Thomas Meyer, Rafael Peñaloza, and Srdjan Vesic, editors, *Joint Proceedings of the 2nd Workshop on Knowledge Diversity and the 2nd Workshop on Cognitive Aspects of Knowledge Representation co-located with 20th International Conference on Principles of Knowledge Representation and Reasoning (KR 2023), Rhodes, Greece, September 3–4, 2023*, volume 3548 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023.

[32] Federico Sabbatini and Roberta Calegari. Explainable clustering with CREAM. In Pierre Marquis, Tran Cao Son, and Gabriele Kern-Isberner, editors, *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR 2023, Rhodes, Greece, September 2–8, 2023*, pages 593–603, 2023.

[33] Federico Sabbatini and Roberta Calegari. Symbolic knowledge-extraction evaluation metrics: The FiRe score. In Kobi Gal, Ann Nowé, Grzegorz J. Nalepa, Roy Fairstein, and Roxana Radulescu, editors, *ECAI 2023 - 26th European Conference on Artificial Intelligence, September 30 – October 4, 2023, Kraków, Poland – Including 12th Conference on Prestigious Applications of Intelligent Systems (PAIS 2023)*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, pages 2033–2040. IOS Press, 2023.

[34] Federico Sabbatini and Roberta Calegari. Unlocking insights and trust: The value of explainable clustering algorithms for cognitive agents. In Rino Falcone, Cristiano Castelfranchi, Alessandro Sapienza, and Filippo Cantucci, editors, *Proceedings of the 24th Workshop "From Objects to Agents", Roma, Italy, November 6–8, 2023*, volume 3579 of *CEUR Workshop Proceedings*, pages 232–245. CEUR-WS.org, 2023.

[35] Federico Sabbatini and Roberta Calegari. Unveiling opaque predictors via explainable clustering: The CReEPy algorithm. In Guido Boella, Fabio Aurelio D'Asaro, Abeer Dyoub, Laura Gorrieri, Francesca A. Lisi, Chiara Manganini, and Giuseppe Primiero, editors, *Proceedings of the 2nd Workshop on Bias, Ethical AI, Explainability and the role of Logic and Logic Programming co-located with the 22nd International Conference of the Italian Association for Artificial Intelligence (AI*IA 2023), Rome, Italy, November 6, 2023*, volume 3615 of *CEUR Workshop Proceedings*, pages 1–14. CEUR-WS.org, 2023.

[36] Federico Sabbatini and Roberta Calegari. On the evaluation of the symbolic knowledge extracted from black boxes. *AI and Ethics*, 4(1):65–74, January 2024.

[37] Federico Sabbatini, Giovanni Ciatto, Roberta Calegari, and Andrea Omicini. Hypercube-based methods for symbolic knowledge extraction: Towards a unified model. In Angelo Ferrando and Viviana Mascardi, editors, *WOA 2022 – 23rd Workshop "From Objects to Agents"*, volume 3261 of *CEUR Workshop Proceedings*, pages 48–60. Sun SITE Central Europe, RWTH Aachen University, November 2022.

[38] Federico Sabbatini, Giovanni Ciatto, Roberta Calegari, and Andrea Omicini. Symbolic knowledge extraction from opaque ML predictors in PSyKE: Platform design & experiments. *Intelligenza Artificiale*, 16(1):27–48, 2022.

[39] Federico Sabbatini, Giovanni Ciatto, Roberta Calegari, and Andrea Omicini. Towards a unified model for symbolic knowledge extraction with hypercube-based methods. *Intelligenza Artificiale*, 17(1):63–75, 2023.

[40] Federico Sabbatini, Giovanni Ciatto, and Andrea Omicini. GridEx: An algorithm for knowledge extraction from black-box regressors. In Davide Calvaresi, Amro Najjar, Michael Winikoff, and Kary Främling, editors, *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*, pages 18–38. Springer Nature, Basel, Switzerland, 2021.

[41] Federico Sabbatini, Giovanni Ciatto, and Andrea Omicini. Semantic Web-based interoperability for intelligent agents with PSyKE. In Davide Calvaresi, Amro Najjar, Michael Winikoff, and Kary Främling, editors, *Explainable and Transparent AI and Multi-Agent Systems*, volume 13283 of *Lecture Notes in Computer Science*, chapter 8, pages 124–142. Springer, 2022.

[42] Federico Sabbatini and Catia Grimani. Symbolic knowledge extraction from opaque predictors applied to cosmic-ray data gathered with LISA Pathfinder. *Aeronautics and Aerospace Open Access Journal*, 6(3):90–95, 2022.

[43] Federico Sabbatini, Catia Grimani, and Roberta Calegari. Bridging machine learning and diagnostics of the esa lisa space mission with equation discovery via explainable artificial intelligence. *Advances in Space Research*, 74(1):505–517, 2024.

[44] Rudy Setiono, Wee Kheng Leow, and Jacek M. Zurada. Extraction of rules from artificial neural networks for nonlinear regression. *IEEE Transactions on Neural Networks*, 13(3):564–577, 2002.

[45] Maria Teresinha Arns Steiner, Pedro José Steiner Neto, Nei Yoshihiro Soma, Tamio Shimizu, and Júlio Cesar Nievola. Using neural network rule extraction for credit-risk evaluation. *International Journal of Computer Science and Network Security*, 6(5A):6–16, 2006.