Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning
Applications and Systems

Symbolic Knowledge Extraction from Opaque Machine Learning Predictors: GridREx & PEDRO

Federico Sabbatini[1] , **Roberta Calegari**[2]
[1]Dipartimento di Scienze Pure e Applicate (DiSPeA),
Università degli Studi di Urbino "Carlo Bo", Italy
[2]Alma AI—Alma Mater Research Institute for Human-Centered Artificial Intelligence,
ALMA MATER STUDIORUM–Università di Bologna, Italy
f.sabbatini1@campus.uniurb.it, roberta.calegari@unibo.it

Abstract

Procedures aimed at explaining outcomes and behaviour of opaque predictors are becoming more and more essential as machine learning (ML) black-box (BB) models pervade a wide variety of fields and, in particular, critical ones – e.g., medical or financial –, where it is not possible to make decisions on the basis of a blind automatic prediction. A growing number of methods designed to overcome this BB limitation is present in the literature, however some ML tasks are nearly or completely neglected—e.g., regression and clustering. Furthermore, existing techniques may be not applicable in complex real-world scenarios or they can affect the output predictions with undesired artefacts.

In this paper we present the design and the implementation of GridREx, a pedagogical algorithm to extract knowledge from black-box regressors, along with PEDRO, an optimisation procedure to automate the GridREx hyper-parameter tuning phase with better results than manual tuning. We also report the results of our experiments involving the application of GridREx and PEDRO in real case scenarios, including GridREx performance assessment by using as benchmarks other similar state-of-the-art techniques. GridREx proved to be able to give more concise explanations with higher fidelity and predictive capabilities.

1 Introduction & Motivation

Nowadays machine learning (ML) techniques – e.g., artificial neural networks (ANNs) – are among the most widespread tools to face almost any kind of task (Rocha, Papa, and Meira 2012). The learning process consists of the model internal parameter tuning in order to maximise its predictive capability w.r.t. the domain data. Despite the impressive predictive power of ML algorithms even in complex scenarios, the major drawback in the application of these techniques is their *opacity*, intended as their inability to provide any intelligible representation of the acquired knowledge. Opaque models are commonly named *black boxes* (BBs), because they can only represent knowledge in a *sub-symbolic* way. Since there exist critical fields where it is not possible to accept BB predictions or recommendations – for instance, healthcare, finance and law domains as well as areas where decision making may affect human lives in terms of health, wealth and freedom – it is of paramount importance to rely on *explainable* predictions to let humans retain accountability and liability over the decisions they make.

According to Guidotti et al. (2018), different strategies can be exploited to pursue the purpose of explainability. For instance, it is possible to obtain explainable data-driven solutions *only* by using *interpretable* algorithms (Rudin 2019)—such as generalised linear models, decision trees, etc. However, in general this can have repercussions on the predictive performance, as the most effective algorithms are excluded—e.g., ANNs. Another strategy consists of deriving *post-hoc* explanations (Kenny et al. 2021), aimed at reverse-engineering the BB inner operation to make it explicit. In this way data scientists are allowed to adopt prediction-effective (but opaque) algorithms without sacrificing human readability. The extraction algorithm presented in this paper belongs to the latter strategy.

Symbolic knowledge extraction (SKE) is among the most promising means to derive *post-hoc* explanations from sub-symbolic BBs. Its main idea is to build a *symbolic* – and thus interpretable – model that mimics the behaviour (a.k.a., output predictions) of the original BB. Symbols may consist of intelligible knowledge—e.g., lists or trees of *rules* that can be exploited to either derive predictions or to better understand the BB behaviour. SKE has been applied, for instance, to credit-risk evaluation (Baesens et al. 2003; Baesens et al. 2001; Steiner et al. 2006), healthcare (Bologna and Pellegrini 1997; Hayashi, Setiono, and Yoshida 2000), credit card screening (Setiono, Baesens, and Mues 2011), intrusion detection systems (Hofmann, Schmitz, and Sick 2003), and keyword extraction (Azcarraga, Liu, and Setiono 2012).

Despite the large amount of SKE techniques existing in the literature, only few of them are explicitly designed for regression—for instance ITER (Huysmans, Baesens, and Vanthienen 2006), REFANN (Setiono, Leow, and Zurada 2002) and GridEx (Sabbatini, Ciatto, and Omicini 2021). Furthermore, they present several limitations in their applications. REFANN is a *decompositional* (Andrews, Diederich, and Tickle 1995) extraction procedure only applicable to 3-layer ANNs and it requires a prior minimisation (not always feasible without loss of predictive performance) of the number of hidden neurons to simplify the extraction process. It is then poorly suited for modern *deep* neural networks. Conversely, ITER and GridEx are *pedagogical* (An-

drews, Diederich, and Tickle 1995) approaches which can be applied to any kind of BB regressor since they do not make assumptions on the type and structure of the underlying model. However, ITER predictive performance degrades when applied to *high-dimensional* data sets. GridEx extends ITER in order to overcome its major drawbacks, but both algorithms produce rule lists where each rule is associated to a constant value, so they introduce an undesired discretisation in the output predictions.

In this work we propose GridREx, a new pedagogical knowledge-extraction procedure extending GridEx to overcome the output discretisation issue. While doing so, GridREx is able to produce more concise rule lists with better predictive performances and fidelity w.r.t. the data and the underlying model predictions, respectively. We demonstrate the effectiveness of GridREx by reporting our experiments on real-world scenarios and comparing its performance with the the most exploited state-of-the-art techniques – namely, ITER, GridEx and CART (Breiman et al. 1984) – trained on the same data. Furthermore, since GridREx – as many other SKE algorithms – has several hyper-parameters to be configured to obtain the best output and since the manual tuning of these parameters can be often tedious and unsatisfactory, we also discuss PEDRO, an optimiser explicitly designed to automate the selection of the best GridREx hyper-parameter values. PEDRO is designed according to the hyper-parameters optimisation methodology widely discussed in the literature (Yang and Shami 2020; Feurer and Hutter 2019; Hutter, Kotthoff, and Vanschoren 2019) targeting progressive automation of ML, based on principles from optimization and machine learning itself.

It is worth mentioning that the experiments reported in Section 5 – aimed at producing logic rules as for output – are grouped into two sections. In the first group we report the results of GridREx trained by using as target values the BB predictor outputs instead of the data set original variable. The performance of GridREx is then compared with that of analogous techniques. In the second group we demonstrate how, under some particular circumstances, GridREx can be applied *directly* to the initial training set as well, to obtain an equivalent model performing *rule induction* (Grzymala-Busse 2005). Although this hypothesis requires further investigation, from our first experiments it appears that it is convenient to exploit the predictor when dealing with scattered training sets. More precisely, a sparse data set can be increased by creating a certain number of new random samples and by using the BB predictor as an oracle to obtain corresponding output values. In this way it is possible to augment the data set and thus produce fewer and more precise rules.

2 Related Works

To ease the reader, in the following we provide a brief description of the knowledge-extraction techniques adopted in the experiments presented in Section 5.

ITER ITER is a pedagogical technique based on the iterative creation and expansion of hypercubes in the input feature space. The algorithm terminates if one of the following conditions holds: *(i)* the input space is entirely covered; *(ii)* it is not possible to expand/add further cubes; *(iii)* the maximum number of iterations is reached. Output rules are produced by creating a certain amount of random samples into each cube, then by using the underlying BB to predict the corresponding output values and finally by averaging the predictions to obtain the mean cube output. Thus, each cube corresponds to a human-comprehensible rule and each rule is associated with a constant output.

GridEx GridEx extends the core concepts of ITER to avoid issues concerning computational time, exhaustivity and fidelity. As a consequence, GridEx can be applied under the same circumstances as ITER, but it returns better results in terms of output rule amount and predictive accuracy. GridEx follows a completely different strategy to create the hypercubes associated with the output rules. It iteratively partitions the input space according to given strategies and a sensitivity threshold parameter. As ITER, it outputs rules associated with constant values.

CART CART is a pedagogical algorithm able to induce a regression tree on a data set or a trained BB. The extraction of human-readable rules is based on the conversion of each possible path from the root to leaves into an *if-then* rule. Also in this case the output value is a constant.

3 GridREx

GridREx – Grid Regressive Extractor – is a pedagogical knowledge-extraction method designed for regression tasks. Consequently, it can be applied to any kind of machine learning predictor if the output variable is numeric. GridREx also requires the input attributes to be numeric, so it cannot be applied if the problem domain is described by relevant categorical variables. The algorithm is designed only for univariate regression, but it is possible to split a multivariate regression task into several univariate tasks and then apply the extractor to each one.

The basic idea behind GridREx is to partition the input feature space into hypercubes, by observing the input instances of the data set and the output predictions of the underlying BB. The algorithm leverages on the assumption according to which close input samples should exhibit similar outputs, or at least output values following a similar regression law w.r.t. the input variables. After having identified these hypercubic clusters of similar data points, GridREx creates a human-readable rule for each one. Output rules are *first-order logic* rules, where the premise is a conjunction of conditions on the input variables – i.e., each variable should belong to a specific interval of values – and the action is a regression rule describing the output variable in terms of a linear combination of the input attributes.

3.1 Hyper-Parameters

The behaviour of the GridREx algorithm can be controlled through several user-defined hyper-parameters. These parameters define how to perform the input space partitioning and thus the amount and quality of the output rules. More precisely, the algorithm hyper-parameters are:

1. $n \in \mathbb{N}_{>0}$, i.e., the number of iterations to perform;
2. $m \in \mathbb{N}_{>0}$, i.e., the minimum number of samples to consider in each hypercube;
3. $\theta \in \mathbb{R}_{>0}$, i.e., the hypercube predictive error threshold;
4. $S_1, \ldots, S_n$, i.e., the splitting strategy to adopt at each algorithm iteration $i, i = 1, \ldots, n$.

The notion of *hypercube predictive error* refers to the mean absolute error between the output value provided by GridREx and that of the underlying BB for all the training samples belonging to a hypercube. The threshold parameter can be exploited by users to set the preferred trade-off between *quality* and *readability* of the output rules—intended as average predictive capability of the hypercubes and number of rules, respectively. Please notice that the more the quality increases, the more the readability decreases.

As for the splitting strategies, GridREx allows users to select a different criterion for each iteration by choosing one of the following:

1. a fixed strategy, parametrised with an integer value k, performing k slices along each input dimension, and
2. an adaptive strategy, parametrised with an increasingly-monotone function of the form $f : [0, 1] \rightarrow \mathbb{N}_{>0}$, performing $f(r)$ slices along each input dimension with *relevance* equal to r.

For feature relevance, we intend numeric information about how an input feature is important to determine the output variable. GridREx assumes that this measure is normalised in the $[0, 1]$ interval, with the most relevant feature having a relevance equal to 1. Importance values can be estimated in several ways—e.g., (Zien et al. 2009; Zhuang et al. 2019; Altmann et al. 2010). Since we have implemented our algorithm in Python, we exploited a simple feature selection method available within the SciKit-Learn Python library: `feature_selection.f_regression`[1].

Experimental results showed that fixed strategies are a good choice when dealing with data sets described by input attributes having comparable relevance. When this condition does not hold, it is possible to reduce the number of partitions along the less relevant dimensions in order to have fewer output rules without noticeable degradation of their quality. It is worthwhile to notice that the number of partitions greatly impacts on the GridREx performances, in terms of efficiency as well as predictive capabilities and output rules readability. In other words, a large number of partitions implies the same amount of rules, so – despite being more accurate – these rules cover smaller input space regions w.r.t. those produced with fewer partitions. As concerns human readability, the output rule set readability decreases as its dimension grows. Accordingly – considering also that the output variable can depend more on some variables and almost not depend at all on others – we develop adaptive strategies allowing GridREx to create more slices on relevant dimensions and fewer partitions on the others, without losses in the extracted rules' quality. In the following, we refer to the term *grid* to identify an object incapsulating both the maximum number of GridREx iterations and the strategy to adopt at each iteration.

[1]cf. https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_regression.html

3.2 Algorithm

The GridREx algorithm, reported in Algorithm 1, requires a data set of training samples – e.g., the one adopted to train the underlying model – and a BB regressor to be used as an *oracle*. This basic version of GridREx can be modified to perform rule induction from data instead of BB explanation. It is sufficient to use the original output of the data set instances as target during the training phase. Conversely, when adopted as knowledge extractor, GridREx firstly substitutes the output values of the training data set with the corresponding underlying BB model predictions.

The main idea behind GridREx is to partition the *surrounding hypercube* – i.e., the minimal region including all the input instances – in a number of (hyper)cubic regions such that the output values of all the samples in each partition can be described with the same linear combination of the input variables. At the end of the partitioning, each hypercube is converted into an *if-then* logical rule.

GridREx adopts a top-down recursive partitioning, starting from a single region equal to the surrounding hypercube. At each iteration, all the eligible regions are split as specified in the user-defined list of strategies. A region is *eligible* if it contains input samples and its error is above the user-defined threshold. Otherwise, regions can be *negligible* if they contain no input samples or *permanent* if their error is smaller than the threshold. The former are discarded, so there will be no rules associated with them, whereas the latter will be no further partitioned. If a partition contains not enough training instances to provide accurate predictions, GridREx generates random samples, using the underlying BB as an oracle to predict the corresponding output values. The minimum number of samples to be present in each hypercube is one of the hyper-parameters described in the previous section. Obviously, this augmentation phase cannot be performed when GridREx is applied for rule induction.

After every splitting iteration a merging phase is performed to reduce the number of eligible regions. GridREx attempts to pair-wise merge adjacent hypercubes containing samples having similar behaviours, on the basis of the threshold parameter. Specifically, two adjacent hypercubes are joined only if the merged cube predictive error does not exceed the threshold, in order not to hinder the resulting rule quality.

GridREx stopping criteria are the absence of eligible regions to split and the reaching of the user-defined maximum number of iterations. If at least one of these two conditions holds, the algorithm terminates.

To calculate the predictive error corresponding to the identified regions as well as to associate a rule to each one of

Algorithm 1 GridREx pseudocode

Require: grid G, predictive error threshold θ, minimum number of samples per hypercube m to be provided

function GRIDREX(R, D)
    $H_0 \leftarrow$ SURROUNDINGCUBE(D)
    return SPLIT(1, H_0, R, D)

function SURROUNDINGCUBE(D)
    return the minimal hypercube including all the samples of D

function SPLIT(i, H, R, D)
    $d \leftarrow$ depth of G, $\quad \Pi \leftarrow \varnothing, \quad \Pi' \leftarrow \varnothing$
    if $i > d$ **then**
        return $\{H\}$
    for all $H' \in$ PARTITIONS(H, i) s.t. $H' \cap D \neq \varnothing$ **do**
        $D \leftarrow D \cup$ GENERATESAMPLESIN(H', D)
        if PREDICTIVEERROR(H', R, D) $\leq \theta$ **then**
            $\Pi \leftarrow \Pi \cup \{H'\}$
        else
            $\Pi' \leftarrow \Pi' \cup \{H'\}$
    $\Pi'' \leftarrow$ MERGE(Π, R, D)
    for all $H' \in \Pi'$ **do**
        $\Pi'' \leftarrow \Pi'' \cup$ SPLIT($i+1, H', R, D$) ▷ Recursive step
    return Π''

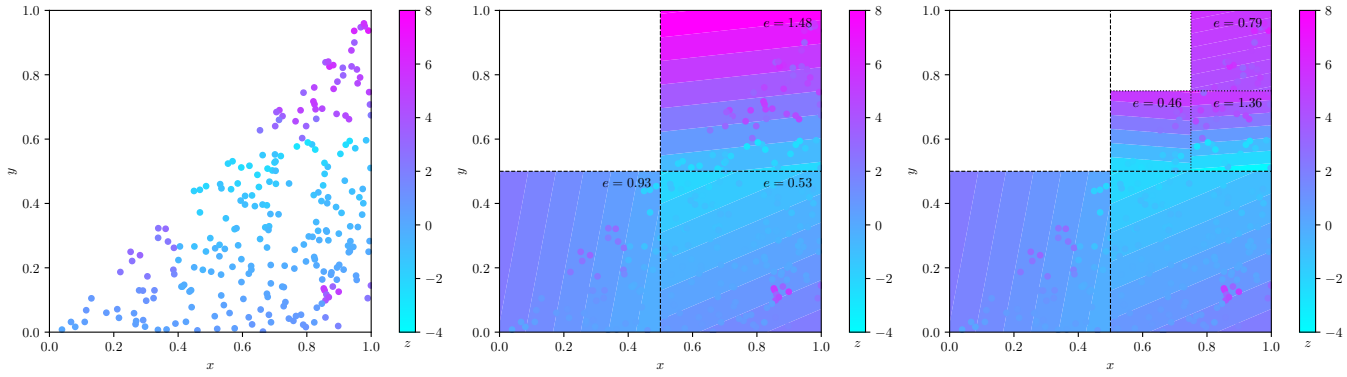
function PARTITIONS(H, i)
    return { all the partitions of H according to the i-th level of the grid G }

function GENERATESAMPLESIN(H, D)
    $c \leftarrow |H \cap D|$
    if $c < m$ **then**
        return { $(m - c)$ random points in H }
    else
        return $\varnothing$

function PREDICTIVEERROR(H, R, D)
    return the predictive error of R w.r.t. the samples of D included in H

function MERGE(Π, R, D)
    $C \leftarrow$ ADJACENTCUBES(Π)
    while ($|C| > 0$) **do**
        $(H_1^*, H_2^*) \leftarrow \underset{(H_1,H_2)\in C}{\arg\min}$ { PREDICTIVEERROR($H_1 \cup H_2, R, D$) }
        $H \leftarrow H_1^* \cup H_2^*$
        if PREDICTIVEERROR(H, R, D) $\leq \theta$ **then**
            $\Pi \leftarrow \Pi \setminus \{H_1^*, H_2^*\} \cup \{H\}$
            $C \leftarrow$ ADJACENTCUBES(Π)
        else
            return Π
    return Π

function ADJACENTCUBES(Π)
    return { $(H_1, H_2) \mid H_1, H_2 \in \Pi \wedge H_1 \neq H_2 \wedge (H_1$ and H_2 are adjacent) }



(a) Surrounding cube. (b) Iteration 1. (c) Iteration 2.

Figure 1: Example of GridREx partitioning (without merging step) performed with fixed strategies ($k = 2$) and threshold equal to 1.

them, a different linear model is trained for each hypercube. As for the evaluation of the feature importance, we relied on a simple model available within the SciKit-Learn Python library: `linear_model.LinearRegression`[2]. The predictions of these linear models are compared with the training target values (the underlying BB predictions or the data set output values) to calculate the error of each region, while output rules are created by reading the estimated coefficients of the linear models. To make predictions, GridREx accepts input data belonging to the surrounding hypercube and outputs the response of the linear models corresponding to the hypercubes including the provided data.

In Figure 1 is reported an example of partitioning performed by GridREx on a 2-dimensional data set by using for each iteration a fixed strategy with $k = 2$. The Figure shows the first 2 iterations and the threshold parameter value is set to 1. Figure 1a represents the surrounding hypercube with the data set samples. The first iteration (see Figure 1b) produces 4 hypercubes. One is negligible (white background) since it contains no training samples. Inside the other three regions different linear models are trained and the corresponding predictive errors are calculated on the training samples and reported on the top right corner of each hyper-cube. By comparing these errors with the threshold it can be noticed that the bottom cubes are permanent, so they will not be further partitioned during successive iterations. Conversely, the top right region error exceeds the threshold, so it is partitioned into 4 smaller cubes (see Figure 1c) in a recursive fashion.

4 PEDRO

The tuning of GridREx hyper-parameters may be quite challenging if performed manually. For such a reason we developed PEDRO, a Proficient and Easy Direct Reliable Optimiser that helps users to find the best values for the critical hyper-parameters.

Among all the GridREx parameters, the minimum number of examples to consider in each hypercube is the only one with negligible impact on the algorithm performance, so PEDRO does not focus on this one. The main idea behind the optimiser is to compare GridREx executions differing only for one hyper-parameter, according to some *ad hoc defined* metrics evaluating both the rule quality enhancements and the readability losses. A (small) worsening in the final rule set readability is acceptable only when it is balanced by a (large enough) enhancement in the average predictive performance (details in the next subsection). This is achieved by iteratively exploring the hyper-parameter space in several directions until the trade-off between the two metrics is no more favourable. Then, after having selected a set of candidate optimal solutions, different scoring metrics are adopted to select the best values for the GridREx parameters in terms of best readability, best predictive performance and best trade-off between the two requirements.

PEDRO is based on the following general assumptions about the behaviour of GridREx:

1. a small (large) threshold produces a large (small) number of small (large) partitions with small (large) predictive error, thus hindering readability in favour of rule quality;
2. analogously, an extra iteration produces (a possibly very high number of) smaller regions with smaller average error;
3. the optimisation *desideratum* is the minimisation of both the number of rules and the corresponding predictive error, so an adequate scoring function should be multiplicative w.r.t. both quantities conveniently weighted.

These assumptions are encoded in the following equations—please notice that it is desirable to have large values for the improvement indices. The improvement between two GridREx executions (G_1, G_2) having growing thresholds ($\theta_1 < \theta_2$; Item 1) is calculated as:

$$thresholdImprovement = 1 - \frac{r_2}{r_1} + \frac{E_1}{E_2}, \qquad (1)$$

being r_i and E_i the number of extracted rules and the predictive error, respectively, associated to the i-th GridREx execution G_i. Differently, the improvement between two

[2] cf. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

GridREx executions (G_1, G_2) having growing maximum iteration values ($n_1 < n_2$; Item 2) is calculated as:

$$depthImprovement = \frac{1}{\left(1 - \frac{E_2}{E_1}\right)^{0.1} \cdot \frac{\lceil r_2 \cdot RTO \rceil}{\lceil r_1 \cdot RTO \rceil}}, \quad (2)$$

where r_i and E_i have the same meaning as before and RTO is the readability trade-off parameter. Finally, the scoring function of each GridREx execution (Item 3) is calculated as:

$$score = E \cdot \lceil 2 \cdot r \cdot RTO \rceil . \quad (3)$$

A good execution has a small score.

4.1 Hyper-Parameters

To exploit PEDRO the following parameters have to be provided to the algorithm:

1. a threshold to stop the iterative research when the predictive error worsens too much;
2. analogously, a threshold to identify when the gain in the model readability is not large enough;
3. a patience parameter to relax the threshold-based stopping criteria, in order to avoid optimal solutions that are only local for a limited parameter space region;
4. a trade-off parameter to define the importance of readability w.r.t. the predictive performance;
5. the maximum number of GridREx iterations to perform for each hyper-parameter combination;
6. the focus of the optimisation—i.e., maximise the fidelity of GridREx w.r.t. the underlying BB predictions or the predictive accuracy w.r.t. the data.

Tuning these parameters is not a hard task, since the maximum number of GridREx iterations is cut off by the thresholding mechanism in the majority of cases and the default threshold values are usually adequate. The patience and trade-off parameters affect the parameter space research and thus the total computational time of the optimisation process. Small (large) patience values imply the premature (delayed) algorithm termination. Small (large) trade-off values tend to disadvantage (privilege) the role of the readability w.r.t. that of the predictive performance when calculating the score of GridREx outputs. The optimisation focus should concern the BB if the goal of the extraction process is to explain the model behaviour or on the data if the goal is to obtain an interpretable and accurate predictor to be used in substitution of the underlying BB. However, with well-performing BBs there is not an appreciable difference between the two choices.

4.2 Algorithm

PEDRO finds the optimal hyper-parameters through iterative research inside the parameter space as reported in Algorithm 2. It finds the optimal GridREx error threshold, the number of iterations it should perform and the splitting strategy to adopt in each iteration. The current implementation of PEDRO assumes that every GridREx iteration adopts the same strategy, that can be fixed or adaptive.

The parameter space exploration starts from 25 different points, corresponding to 25 different splitting strategies (2 fixed and 23 adaptive). This choice is an optimal trade-off between combinatorial exhaustivity and search efficiency. Indeed, fixed strategies performing more than 3 slices per dimension result in an explosion of the number of output rules, hindering readability, so as for fixed strategies testing 2 of them (performing 2 and 3 slices, respectively) is sufficient. On the other hand, 23 adaptive strategies were always more than sufficient in our experiments to cover all the interesting splitting combinations: as a matter of fact, they generally result in duplicate strategies. Since they are chosen by default, it is common to have more than one strategy equivalent to others, depending on the input feature importance of the data set to analyse. When this is the case, equivalent strategies are collapsed. For instance, by taking as an example a regression task with 2 input dimensions, if we define 3 adaptive strategies by imposing a certain number of partitions for features having relevance above 0.3, 0.5 and 0.8, respectively, and the actual feature relevancies are 0.1 and 1, all the strategies will perform the same partitioning, leading to the same output. It is worth noting that strategies, especially when dealing with high-dimensional data sets, can lead to an explosion of the number of rules. For this reason, PEDRO automatically excludes all those strategies performing a number of partitions greater than a given value. Such value is equal to 3 times the number of partitions obtained with a fixed strategy performing 2 slices along each dimension. Additionally, after collapsing duplicated strategies, the average amount of partitions is calculated. This enables more strict patience policies to be adopted for strategies having more partitions than the average since they are supposed to be more time-consuming (due to better readability, this detail is not included in the pseudo-code).

For each strategy and for each possible number of GridREx iterations between 1 and the maximum user-defined value, the best threshold of the extraction procedure parametrised with the particular strategy is selected. An early stopping is performed if the loss in readability is no more balanced with the gain in quality (see Equation (2)).

To find the best threshold for a strategy, given a number of iterations, the following steps are executed. Firstly, an initial threshold t_0 and update step s are set to $0.9e$ and $0.5e$, respectively, being e the mean absolute error of the underlying BB. This choice derives from the consideration according to which it is generally not possible to achieve a significantly better performance than the underlying BB by using an extraction procedure that exploits the outputs of such BB as target values. Then GridREx is performed with growing threshold values, starting from t_0 and incrementing it at each iteration by an amount equal to s. If an iteration results in no improvement, the step value s is augmented. It is worth noticing that a rule amount reduction smaller than the user defined value is not considered an improvement. The search terminates if GridREx produces only one output rule, if the worsening w.r.t. the error obtained with the initial threshold t_0 is beyond the user-defined value, or if there are no improvements even by considering the patience parameter (see Equation (1)).

Algorithm 2 PEDRO pseudocode

Require: parameters $maxDepth$, $maxErrorIncrease$, $minRuleDecrease$, $readTradeoff$, $patience_0$ to be provided

function PEDRO(R, D)
    $maxPartitions \leftarrow$ MAXPARTITIONNUMBER(D)
    $S \leftarrow$ INITIALIZESTRATEGIES($D, maxPartitions$)
    return SEARCH(S, R, D, $maxPartitions$)

function MAXPARTITIONNUMBER(D)
    return an upperbound for the number of partitions calculated w.r.t. the number of input features describing D

function INITIALIZESTRATEGIES(D, $maxPartitions$)
    $S \leftarrow$ a set of default strategies
    return $\{\ s \mid s \in S \wedge \text{NPARTITIONS}(D, s) < maxPartitions\ \}$

function NPARTITIONS(D, $strategy$)
    return the number of partitions produced by $strategy$ applied to D

function SEARCH(S, R, D, $maxPartitions$)
    return $\bigcup_{s \in S}$ DEPTHSEARCH($s, R, D, maxPartitions$)

function DEPTHSEARCH(s, R, D, $maxPartitions$)
    $\Pi \leftarrow \varnothing, \quad i \leftarrow 1, \quad best' \leftarrow$ **undefined**
    while $i < maxDepth$ **do**
        $grid \leftarrow$ GRID(i, s)
        $\Pi' \leftarrow$ THRESHOLDSEARCH($grid, R, D, maxPartitions$)
        $best \leftarrow$ SELECTBEST(Π')
        $\Pi \leftarrow \Pi \cup \Pi'$
        if $(|\Pi| > 1) \wedge$ (IMPROVEMENT($best, best'$, "depth") < 1.2) **then return** Π
        $i \leftarrow i + 1$
        $best' = best$
    return Π

function SELECTBEST(Π)
    return the best output in Π w.r.t. both predictive performance and readability according to $readTradeoff$

function IMPROVEMENT($output_1$, $output_2$, $criterion$) ▷ $criterion$ is either "depth" or "threshold"
    return a numeric evaluation of $output_1$ w.r.t. $output_2$ according to $criterion$

function THRESHOLDSEARCH($grid$, R, D, $maxPartitions$)
    $\theta_0 \leftarrow$ PREDICTIVEERROR(D, R)
    $\Pi \leftarrow \varnothing, \quad \theta \leftarrow 0.9\,\theta_0, \quad patience \leftarrow patience_0, \quad step = 0.5\,\theta_0$
    while $patience > 0$ **do**
        $E \leftarrow$ GRIDREX(R, D)
        $p \leftarrow$ parameters of E
        $n \leftarrow$ number of rules extracted by E
        $e \leftarrow$ PREDICTIVEERROR(D, E)
        if $\Pi = \varnothing$ **then** $e_0 \leftarrow e$
        else if $n = 1$ **then return** $\Pi \cup \{(p, n, e)\}$
        else if $n > maxPartitions$ **then return** Π
        else if $e > e_0 \cdot maxErrorIncrease$ **then return** Π
        else if (IMPROVEMENT($(n, e), (n', e')$, "threshold") $\leq 1) \bigvee (n > \lceil n' \cdot minRuleDecrease \rceil)$ **then**
            $patience \leftarrow patience - 1$
            $step \leftarrow \max\{step,\ |e - \theta|\}\ /\ \max\{patience,\ 1\}$
        $\Pi \leftarrow \Pi \cup \{(p, n, e)\}$
        $n' \leftarrow n$
        $e' \leftarrow e$
        $\theta \leftarrow \theta + step$
    return Π

function PREDICTIVEERROR(D, P) ▷ Predictor P may be a BB or an extractor
    return the predictive error of P applied to the samples of D

At the end of the parameter space search PEDRO ranks each solution according to 3 different dimensions: *(i)* the number of extracted rules, a.k.a., the readability; *(ii)* the predictive error, a.k.a., the quality; *(iii)* the *ad hoc* scoring function explicitly designed to select the best trade-off between readability and quality, involving the user-defined specific parameter (see Equation (3)).

5 Comparison in PSyKE

5.1 Knowledge Extraction

In order to assess the performance of GridREx and PEDRO[3] we applied our algorithms to 6 different real-world data sets from the StairwAI[4] EU project. These data sets are composed uniquely of continuous input and output features. 2 data sets have 5 input attributes, whereas the others have 1 input variable. All data sets count 12 100 instances and we adopted an 80%-20% train-test split. The test set was never used to train the BBs or the extractors.

Our results are summarised in Table 1. We used PEDRO to estimate the best GridREx hyper-parameters for each data set. Other state-of-the-art extraction procedures – namely, ITER, GridEx, and CART – applied on the same data sets are used as benchmarks to compare with GridREx.

We performed a complete comparison by collecting the same indicators for each extractor. For regression tasks, the most used indicators are the mean absolute/squared error (MAE/MSE) and the R^2 score. Due to space limitations, we report in Table 1 only the MAE (aligned to the results of other indicators).For each data set, the number of input variables, the MAE of the BB model applied to it and the results of the 4 extraction procedures applied to the data sets and the corresponding BBs are reported in Table 1. For each extractor, the number of extracted rules (R) and the MAE w.r.t. both the data (D) and the underlying BB are reported as well. Since ITER can be a non-exhaustive procedure, for this algorithm we included also the percentage of missed test predictions (MTP), i.e., how many test samples ITER is not able to predict w.r.t. the size of the test set.

GridREx proved to be the best extractor for all the case studies, being able to provide the fewest number of rules and, at the same time, achieving the highest predictive performance of all the tested extractors w.r.t. both the data and the underlying BB predictions. This is not surprising, since all the other techniques are not able to produce actual regression rules, but only constant values assigned to defined input space regions. This undesired output discretisation hinders their predictive performance as well as their readability, since they may fragment regions dominated by the same linear relationship into a number of smaller regions having constant outputs.

Obviously, the results reported in Table 1 are not the unique possible. The number of rules and the MAE of each procedure may greatly vary with different algorithm parameters. While we used PEDRO to choose the best GridREx and GridEx hyper-parameters, different considerations are mandatory for the other algorithms. We selected for CART a maximum depth equal to 2, in order to have not more than 4 output rules. This choice derives from the fact that GridREx produces 3-5 output rules for all the data sets; in such a way it is possible to compare the predictive performance of CART and GridEx in terms of MAE to parity of human readability. It is worth noticing that CART can achieve a predictive performance almost as good as that of GridREx at the expense of readability—i.e., with larger depths and, therefore, larger rule sets. For instance, by using a maximum depth equal to 3, resulting in 8 output rules, it is possible to have better predictive performance. However, since CART outputs are constant values while those of GridREx are not, the latter is able to outperform the former in the general case.

On the other hand, ITER shows quite good results in terms of MAE when applied to data sets with only one input feature, but it tends to have smaller readability than GridREx and furthermore its rules are always non-exhaustive. For ITER we report here the best results obtained after a manual hyper-parameter tuning phase. As *best* we intend the predictive performance w.r.t. the output model readability.

5.2 Rule Induction

In the following we describe the results of GridREx applied to the same aforementioned data sets without the intermediate "interference" of the BB predictor. These results are summarised in Table 2 as a comparison between the extraction performed by GridREx on the BB regressor and the rule induction directly performed by GridREx on the data sets. We describe the GridREx executions in terms of MAE and R^2 score as predictive error/performance measurement and the number of output rules as readability index.

Our goal is to demonstrate that this procedure can be exploited instead of ML techniques to learn relationships from data without loss of predictive performance. Extraction techniques from BB predictors imply prior training and tuning phases regarding the BB itself, often a time-consuming task that offers no certainty about the optimality of the selected final hyper-parameters. On the other hand, rule induction is not able to rely on an ML oracle to augment the data set under study, nor can learn complex relationships as other widespread ML algorithms do—for instance, ANNs. For these reasons, GridREx should be exploited for this purpose only when a dense data set is available, so as to have the possibility to obtain rules corresponding to hypercubes naturally including a proper amount of samples—used to construct robust and accurate output rules.

From the results reported in Table 2 it is clear how GridREx can be applied to perform rule induction obtaining approximatively the same results as the rule extraction from an ML BB—and, in some cases, even slightly better results. For this reason, we suggest directly applying GridREx to the data set when possible, in order to avoid the training and tuning phases of the underlying regressor. PEDRO can be ex-

[3]Experiments have been run exploiting the Python implementations provided by PSyKE (Sabbatini et al. 2021), cf. https://apice.unibo.it/xwiki/bin/view/PSyKE/ and https://github.com/psykei/psyke-python/

[4]https://cordis.europa.eu/project/id/101017142; data sets will be publicly available accordingly to the timeline of the StairwAI EU project

Data set	Input var.	BB MAE	GridREx R	GridREx MAE D	GridREx MAE BB	ITER R	ITER MAE D	ITER MAE BB	ITER MTP (%)	GridEx R	GridEx MAE D	GridEx MAE BB	CART R	CART MAE D	CART MAE BB
#1	5	0.4	3	1.4	1.5	76	9.5	7.2	6.3	5	14.5	14.6	4	14.6	14.6
#2	1	4.3	5	5.0	3.3	20	6.3	4.3	1.7	5	15.3	14.7	4	17.8	17.5
#3	1	8.3	5	11.3	6.7	14	12.2	7.5	1.8	5	17.6	14.9	4	17.1	12.9
#4	5	1.5	4	22.1	21.9	83	23.2	21.4	9.6	5	28.4	28.3	4	26.2	26.1
#5	1	3.8	5	4.7	1.8	4	4.6	2.0	1.7	3	4.8	2.2	4	4.7	1.9
#6	1	0.8	5	1.0	0.7	15	1.6	1.3	1.9	5	3.2	3.1	4	3.9	3.8

Table 1: Results of GridREx applied to the 6 described data sets in terms of number of extracted rules (R) and MAE w.r.t. the data (D) and the underlying BB model. Results are compared with those of ITER, GridEx and CART applied to the same data sets. For each data set the number of input variables and the BB MAE are reported, as well as the percentage of missed test predictions (MTP) for ITER.

Data set	Input variables	BB MAE (data)	BB R^2 (data)	GridREx (extraction) # Rules	GridREx (extraction) MAE (data)	GridREx (extraction) R^2 (data)	GridREx (extraction) MAE (BB)	GridREx (extraction) R^2 (BB)	GridREx (induction) # Rules	GridREx (induction) MAE (data)	GridREx (induction) R^2 (data)
#1	5	0.4	1.00	3	1.4	1.00	1.5	1.00	3	1.5	1.00
#2	1	4.3	0.99	5	5.0	0.99	3.3	1.00	5	4.6	0.99
#3	1	8.3	0.97	5	11.3	0.92	6.7	0.97	5	11.4	0.91
#4	5	1.5	0.99	4	22.1	0.67	21.9	0.67	5	21.8	0.70
#5	1	3.8	0.28	5	4.7	0.05	1.8	0.19	5	3.91	-0.1
#6	1	0.8	1.00	5	1.0	0.99	0.7	1.00	5	1.9	0.99

Table 2: Comparison in terms of readability, predictive error and R^2 score of GridREx applied to the 6 described data sets by exploiting the underlying BB (knowledge extraction) as well as without any intermediate ML predictor (rule induction).

ploited to detect the best GridREx hyper-parameters also for the case of rule induction. We adopted it to tune GridREx during our rule induction experiments to obtain the results reported in the Table 2.

5.3 Discussion on the Execution Time

Except for CART, the execution time of the compared algorithms strictly depends on the number of extracted rules. In all our experiments, CART always produces its output within a second. The same holds for GridEx and GridREx when less than 10 hyper-cubes are created. These two techniques may require up to one minute when dealing with huge amounts of hyper-cubes—e.g., above 500. ITER results to be slower than the others, due to the creation and discard of many data structures during its execution, as well as for the repetition of the same operations several times. These considerations can be drawn by analysing the algorithm description given by the authors of ITER. In our experiments, ITER terminates within a second only when less than 6 rules are extracted. The execution in the other cases may take up to some minutes, depending on its hyper-parameters.

6 Conclusions

In this paper we present the design and application of GridREx and PEDRO, a pedagogical knowledge-extraction algorithm for BB regressors of any kind and an optimisation procedure applicable to it to help human users during the hyper-parameter tuning phase of GridREx. GridREx is able to overcome at the same time the major limitations of similar algorithms described in the literature—i.e., the inability to output regression rules and the application restricted to specific BBs. PEDRO can provide the best hyper-parameter values for GridREx, taking into account the readability of the output rule set, the corresponding predictive performance, or both of the two criteria, combining them according to a user-provided trade-off threshold. PEDRO outperforms manual tuning executed by human experts even when launched with default parameters.

Even though GridREx proved to outperform several state-of-the-art analogous techniques, some considerations can be taken into account for future improvements. Two examples are *(i)* the possibility to customise the relationship between input and output inside single hypercubes – such as polynomial functions instead of linear ones – and *(ii)* the introduction of a splitting technique disentangled from the equal-size partitioning criterion, possibly critical when dealing with strongly asymmetrical data sets.

In the future we also plan to enrich PEDRO, making it more configurable by users. For example, we find it useful to let users choose among more than one criterion to measure the predictive error – or, more generally, performance – of a given regressor/extractor—for instance, MSE and R^2 score, as well as user-defined loss functions. We also plan to modify the search of the input parameter space to include grids having not only one single strategy for every iteration, but also different strategies automatically selected by PEDRO.

Acknowledgements

This work has been partially supported by the EU ICT-48 2020 project TAILOR (No. 952215) and by the European Union's Horizon 2020 research and innovation programme under G.A. no. 101017142 (StairwAI project).

References

Altmann, A.; Toloşi, L.; Sander, O.; and Lengauer, T. 2010. Permutation importance: a corrected feature importance measure. *Bioinformatics* 26(10):1340–1347.

Andrews, R.; Diederich, J.; and Tickle, A. B. 1995. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems* 8(6):373–389.

Azcarraga, A.; Liu, M. D.; and Setiono, R. 2012. Keyword extraction using backpropagation neural networks and rule extraction. In *The 2012 International Joint Conference on Neural Networks (IJCNN 2012)*, 1–7. IEEE.

Baesens, B.; Setiono, R.; De Lille, V.; Viaene, S.; and Vanthienen, J. 2001. Building credit-risk evaluation expert systems using neural network rule extraction and decision tables. In Storey, V. C.; Sarkar, S.; and DeGross, J. I., eds., *ICIS 2001 Proceedings*, 159–168. Association for Information Systems.

Baesens, B.; Setiono, R.; Mues, C.; and Vanthienen, J. 2003. Using neural network rule extraction and decision tables for credit-risk evaluation. *Management Science* 49(3):312–329.

Bologna, G., and Pellegrini, C. 1997. Three medical examples in neural network rule extraction. *Physica Medica* 13:183–187.

Breiman, L.; Friedman, J.; Stone, C. J.; and Olshen, R. A. 1984. *Classification and Regression Trees*. CRC Press.

Feurer, M., and Hutter, F. 2019. Hyperparameter optimization. In *Automated machine learning*. Springer, Cham. 3–33.

Grzymala-Busse, J. W. 2005. Rule induction. In Maimon, O., and Rokach, L., eds., *The Data Mining and Knowledge Discovery Handbook*. Springer. 277–294.

Guidotti, R.; Monreale, A.; Ruggieri, S.; Turini, F.; Giannotti, F.; and Pedreschi, D. 2018. A survey of methods for explaining black box models. *ACM Computing Surveys* 51(5):1–42.

Hayashi, Y.; Setiono, R.; and Yoshida, K. 2000. A comparison between two neural network rule extraction techniques for the diagnosis of hepatobiliary disorders. *Artificial intelligence in Medicine* 20(3):205–216.

Hofmann, A.; Schmitz, C.; and Sick, B. 2003. Rule extraction from neural networks for intrusion detection in computer networks. In *2003 IEEE International Conference on Systems, Man and Cybernetics*, volume 2, 1259–1265. IEEE.

Hutter, F.; Kotthoff, L.; and Vanschoren, J. 2019. *Automated machine learning: methods, systems, challenges*. Springer Nature.

Huysmans, J.; Baesens, B.; and Vanthienen, J. 2006. ITER: An algorithm for predictive regression rule extraction. In *Data Warehousing and Knowledge Discovery (DaWaK 2006)*, 270–279. Springer.

Kenny, E. M.; Ford, C.; Quinn, M.; and Keane, M. T. 2021. 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.

Rocha, A.; Papa, J. P.; and Meira, L. A. A. 2012. How far do we get using machine learning black-boxes? *International Journal of Pattern Recognition and Artificial Intelligence* 26(02):1261001–(1–23).

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

Sabbatini, F.; Ciatto, G.; Calegari, R.; and Omicini, A. 2021. On the design of PSyKE: A platform for symbolic knowledge extraction. In Calegari, R.; Ciatto, G.; Denti, E.; Omicini, A.; and Sartor, G., eds., *WOA 2021 – 22nd Workshop "From Objects to Agents"*, volume 2963 of *CEUR Workshop Proceedings*, 29–48. Sun SITE Central Europe, RWTH Aachen University. 22nd Workshop "From Objects to Agents" (WOA 2021), Bologna, Italy, 1–3 September 2021. Proceedings.

Sabbatini, F.; Ciatto, G.; and Omicini, A. 2021. GridEx: An algorithm for knowledge extraction from black-box regressors. In Calvaresi, D.; Najjar, A.; Winikoff, M.; and Främling, K., 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*. Basel, Switzerland: Springer Nature. 18–38.

Setiono, R.; Baesens, B.; and Mues, C. 2011. Rule extraction from minimal neural networks for credit card screening. *International Journal of Neural Systems* 21(04):265–276.

Setiono, R.; Leow, W. K.; and Zurada, J. M. 2002. Extraction of rules from artificial neural networks for nonlinear regression. *IEEE Transactions on Neural Networks* 13(3):564–577.

Steiner, M. T. A.; Steiner Neto, P. J.; Soma, N. Y.; Shimizu, T.; and Nievola, J. C. 2006. Using neural network rule extraction for credit-risk evaluation. *International Journal of Computer Science and Network Security* 6(5A):6–16.

Yang, L., and Shami, A. 2020. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing* 415:295–316.

Zhuang, J.; Dvornek, N. C.; Li, X.; Yang, J.; and Duncan, J. 2019. Decision explanation and feature importance for invertible networks. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, 4235–4239. IEEE.

Zien, A.; Krämer, N.; Sonnenburg, S.; and Rätsch, G. 2009. The feature importance ranking measure. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2009)*, 694–709. Springer.