GridEx: An Algorithm for Knowledge Extraction from Black-Box Regressors

Federico Sabbatini(✉), Giovanni Ciatto, and Andrea Omicini

Dipartimento di Informatica – Scienza e Ingegneria (DISI), Alma Mater Studiorum—Università di Bologna, Cesena, Italy
{f.sabbatini,giovanni.ciatto,andrea.omicini}@unibo.it

Abstract. Knowledge-extraction methods are applied to ML-based predictors to attain explainable representations of their operation when the lack of interpretable results constitutes a problem. Several algorithms have been proposed for knowledge extraction, mostly focusing on the extraction of either lists or trees of *rules*. Yet, most of them only support supervised learning – and, in particular, *classification* – tasks. ITER is among the few rule-extraction methods capable of extracting symbolic rules out of sub-symbolic regressors. However, its performance – here intended as the *interpretability* of the rules it extracts – easily degrades as the complexity of the regression task at hand increases.

In this paper we propose GridEx, an extension of the ITER algorithm, aimed at extracting symbolic knowledge – in the form of lists of if-then-else rules – from *any* sort of sub-symbolic regressor—there including neural networks of arbitrary depth. With respect to ITER, GridEx produces *shorter* rule lists retaining higher fidelity w.r.t. the original regressor. We report several experiments assessing GridEx performance against ITER and CART (i.e., decision-tree regressors) used as benchmarks.

Keywords: Explainable AI · Knowledge extraction · Interpretable prediction · Regression · ITER · GridEx

1 Introduction

Nowadays, black-box data-driven predictors such as neural networks or support-vector machines are among the most used tools to solve a wide range of different tasks [35]. Such predictors are *opaque* systems that operate in a sub-symbolic fashion, making it very hard for humans to understand how they manipulate data to compute their outputs. Nevertheless, they are being increasingly adopted in many application fields – including, but not limited to, healthcare, finance, and law – to support forecasting and decision making.

Thus, to enable the exploitation of black-box predictors within critical applications where interpretability is not an option, several methods have been developed to extract *intelligible* knowledge out of black-box predictors [4], aimed at *explaining* the operation and the outcomes of black boxes to humans.

© Springer Nature Switzerland AG 2021
D. Calvaresi et al. (Eds.): EXTRAAMAS 2021, LNAI 12688, pp. 18–38, 2021.
https://doi.org/10.1007/978-3-030-82017-6_2

Virtually all knowledge-extraction methods proposed so far focus on the extraction of either lists or trees of *rules*, and are exploited in many application areas. For instance, knowledge extraction is applied to credit-risk evaluation [6,7,41]. In healthcare, they are used to make early breast cancer prognosis predictions [21], to help the diagnosis of hepatobiliary disorders [25], coronary artery disease or thyroid dysfunctions [10], to determine the type of dermatological diseases, and to discriminate among liver diseases or diabetes [9]. Rule-extraction algorithms are also applied to, e.g., predictive models for credit card screening [39], intrusion detection in computer networks [26], and keyword extraction [5].

However, while most of the algorithms from the literature focus on *classification* tasks – e.g. Trepan [19], Rule-extraction-as-learning [18], and others [8,31] –, a few are explicitly designed for *regression* tasks—such as Iter [27] and RefAnn [40]. To the best of our knowledge, no algorithm has been proposed so far to tackle other branches of machine learning, such as unsupervised learning. RefAnn is a *decompositional* extraction procedure which can be only applied to neural networks with just *one* hidden layer, and also requires a reduction of the network aimed at minimising the number of hidden neurons, to simplify the extraction process. It is then poorly suited for modern *deep* neural networks. Conversely, Iter is a *pedagogical* approach [3] which can be applied to regressors of any sort, as it does not make any assumption on the type, structure, and operation of the regressors it is applied to. However, its predictive performance degrades when applied to *high-dimensional* data sets.

Accordingly, in this work we propose GridEx, a new knowledge-extraction procedure extending the Iter algorithm to overcome its limitations and to reduce its computational-time complexity. As an extension of Iter, GridEx inherits a number of relevant features. For instance, they both extract rule lists out of regressors of *any* sort. However, GridEx outperforms Iter in terms of *fidelity* of the extracted rules w.r.t. the underlying regressor, especially when applied to high-dimensional data sets. In other words, GridEx extracts rule lists whose predictive capabilities are generally closer to the original black box.

To demonstrate the effectiveness of GridEx, we present a number of experimental evaluations aimed at comparing GridEx and Iter. The predictions of both extraction algorithms are compared among each other and w.r.t. a decision tree regressor (Cart) trained on the same data. This evaluation is repeated on six data sets – having incremental amounts of dimensions and instances –, in order to analyse GridEx scalability, other than predictive performance and its ability to mimic – and therefore explain – the underlying black-box predictor.

2 State of the Art

As the adoption of machine-learning (ML) predictors pervades human activities, critical aspects become more evident and challenging, and require more care. In particular, as widely recognised within the explainable AI (XAI) community, the exploitation of ML comes at the price of relying on *sub-symbolic* algorithms that leverage on poorly-intelligible mechanisms for their operation, since they do not represent knowledge explicitly. Lacking *interpretability*, those algorithms – such as artificial neural networks (ANNs) and support vector machines (SVMs) – are often described as "black boxes" [30]. While it may be negligible or harmless in some application scenarios, interpretability is a critical issue in a growing number of areas. Several solutions have been proposed in the XAI field: the exploitation of (more) interpretable predictors – such as linear models and decision trees – rather than (more) opaque ones – e.g., ANNs and SVMs – in the particular case of supervised learning [36]; or, the exploitation of inspection techniques focusing on either input/output or the black-box internal structure [24].

As discussed in [16], computational systems can be considered as *interpretable* if their operation and outcomes can be easily understood by a human being: unfortunately, most predictors exploited in modern AI tend to sacrifice interpretability, by becoming increasingly complex while seeking for predictive performance. Instead, this paper focuses on those techniques aimed at explaining a sub-symbolic predictor *ex-post*. We restrict our scope to *knowledge-extraction* algorithms which attempt to explain black-box predictors by reverse-engineering their machinery, with the purpose of making their knowledge explicit.

2.1 Knowledge Extraction

Within the scope of supervised learning, knowledge extraction refers to the task of extracting some explicit intelligible representation for the sub-symbolic knowledge acquired by some predictor (either classifier or regressor) via learning from data. Assuming that a procedure for knowledge extraction exists for a particular predictor, any extracted knowledge can then be used as a basis to construct *explanations* – and sometimes as a replacement – for that predictor, provided that such knowledge retains a high *fidelity* w.r.t. the original predictor and the data it has been trained upon [15]. Extracted knowledge, in turn, may then enable further manipulations for the user's benefit—such as merging the knowledge of two or more black-boxes [14]. Unfortunately, no one-size-fit-all solution exists for this task, and several algorithms have been proposed for this purpose [24].

According to [13], virtually all knowledge-extraction methods proposed so far into the literature can be categorised along three orthogonal dimensions, namely: *(i)* the supported sort of learning tasks, *(ii)* the form of the knowledge extracted, and *(iii)* the *translucency* requirements of the black box.

Item *(i)* refers to which supervised learning tasks must be supported by a black box to enable extraction. While most methods support *classification* tasks – e.g. Trepan [19], Rule-extraction-as-learning [18] and others [8,31] –, only a few are explicitly designed to tackle *regression* tasks—such as Iter [27], RefAnn [40], Ann-DT [38] and RN2 [37]. Methods extracting knowledge from black boxes independently of their task are, e.g., G-Rex [29] and Cart [11].

Conversely, item *(ii)* refers to the form of the extracted knowledge. As decision rules [22,28,32] and decision trees [33,34] are the most widespread human-understandable predictors, most methods produce either decision rules or trees.

Finally, the translucency notion [3] from item *(iii)* refers to the relationship between the extracted rules and the *internal* structure of the underlying black box—and how much of it the extraction procedure can take into account. In particular, there exist two sorts of knowledge extractors w.r.t. translucency. *Decompositional* extractors take into account the black-box internal structure during the extraction process, whereas *pedagogical* ones do not. Therefore, pedagogical approaches are usually more general, despite potentially less precise.

To evaluate the quality of knowledge-extraction methods, different indicators can be exploited, including fidelity and predictive performance measurements [42]. In particular, the former is a meta-measure of how good the extracted knowledge mimics the underlying black-box predictions. The latter measures the predictive power of the explanator with respect to the data. In both cases measurements are taken via the same scoring function used for assessing the performance of the black box—which in turn depends on the black-box performed task. For instance, in the particular case of black-box regressors, the mean absolute error (MAE) and the R^2 scores could be exploited.

2.2 The Iter Algorithm

Iter [27] 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 regressors.

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 *surrounding hypercube* containing the whole data set following a bottom-up strategy.

At the end of the process, each partition is converted into a rule of the form

$$\begin{aligned}&\text{if } \mathit{Var_1} \in [\mathit{Value_1}^{\mathit{Low}}, \mathit{Value_1}^{\mathit{High}}]\\ &\text{and } \mathit{Var_2} \in [\mathit{Value_2}^{\mathit{Low}}, \mathit{Value_2}^{\mathit{High}}]\\ &\text{and ... and } \mathit{Var_k} \in [\mathit{Value_k}^{\mathit{Low}}, \mathit{Value_k}^{\mathit{High}}]\\ &\text{then predict some } \mathit{Constant}\end{aligned}$$

where k is the dimension of the input space, i.e. the number of input variables. The predicted output value – *Constant* – is attained by averaging the output values of all samples belonging to the originating hypercube. To compute *Constant* for each hypercube, samples can be both picked from the data set or randomly generated. In the latter case, the underlying regressor is used as an *oracle*.

Pros and Cons. As a pedagogical approach, Iter supports any sort of black-box regressor. For instance, it can be applied to ANNs with any number of hidden layers and neurons, unlike decompositional algorithms as RefAnn.

The if-then-else rules produced by Iter are human-readable and *globally* approximate the underlying black box with high fidelity. Therefore, when the total amount of hypercubes – and rules – found by ITER is relatively small, the resulting rule list is a valuable form of explanation for the underlying black box.

As for the predictive performance, the authors of ITER report very good results with respect to both the data set samples and the underlying black-box outputs. However, the performance of ITER easily degrades when the algorithm is applied to complex data sets—where the complexity is represented by the number of input dimensions. Furthermore, several major ITER drawbacks are also reported, such as *(i)* non-exhaustivity – i.e. the extracted rules do not cover the entire input space –, described with more details in Sect. 2.2 and shown in Fig. 1, *(ii)* the impossibility to handle categorical features, and *(iii)* the impossibility to associate anything than a constant value to each rule—which introduces an undesired discretisation in the predicted values.

In our experience, another limitation of ITER concerns the hypercube expansion mechanism. In fact, as further discussed in Sect. 4, the algorithm may waste a lot of computational efforts processing *irrelevant* regions of the input space – i.e. regions containing no samples from the data set –, and therefore ending up producing several useless rules—which, ultimately, hinder interpretability.

Non-exhaustivity Issue. ITER's hypercube expansion is an iterative procedure strongly affected by the initial conditions, such as the number, position and dimension of the starting cubes. Even by tuning the algorithm parameters, there is always a chance that hypercupe expansion converges to a situation where some portions of the input space are left uncovered. This is undesirable, since the rule list resulting by ITER would then be poorly predictive for data laying in those portions of the input space.

To better clarify the issue, we report in Fig. 1 a trivial example with only two input features taken from [27]. There, the authors show how, after several iterations, the further expansion of the cubes is impossible and a little region of the input space – namely, the central one – is not covered by any of them.

To circumvent this issue, the same authors suggest the creation of additional, smaller cubes to fill the uncovered area. However, despite this solution provides good results for simple data sets, the opposite is true in more complex contexts. In these cases, the uncovered regions require an high number of small hypercubes to be generated. They are smaller and smaller, resulting in an explosion of the amount of rules—which would in turn hinder the interpretability of the final rule list.

The authors suggest another fix for the non-exhaustivity issue: adopting a smaller hypercube update parameter. However, this solution implies more iterations and longer execution time. Thus, this may increase the chance of the algorithm terminating without converging to a valuable partitioning within the maximum iterations limit.

3 GridEx

The design of GridEx aims at overcoming the non-exhaustivity of ITER, other than its inability to discriminate among *interesting* and *negligible* regions of

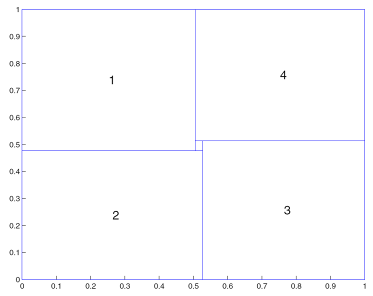


Fig. 1. Example of the Iter non-exhaustivity taken from [27].

the input space. We consider as interesting the regions that contain at least one training-set sample, with the others considered as negligible. Furthermore, GridEx is designed to tackle *complex* data sets—i.e. high-dimensional data sets whose data distribution is non-trivial. In particular, the goal of GridEx is to find bigger and more interesting regions than Iter, while retaining the idea that samples belonging to the same region should have a similar output value. In doing so, GridEx tries to keep the computational and human efforts minimal—where by "human effort" we mean manual parameter tuning, whereas by "computational effort" we mean time and memory requirements for the algorithm execution.

3.1 The Algorithm

GridEx is a pedagogical knowledge-extraction algorithm that – similarly to Iter – produces rule lists out of sub-symbolic predictors, by using them as oracles. In other words, GridEx only takes into account the inputs and outputs of the underlying predictor. For this reason its performance is not tied to the kind or the structure of the model and it can thus be applied to neural networks regardless of their depth, as well as to other sorts of regressors.

Similarly to Iter, GridEx assumes that a black-box regressor R is available, as well as the input data D it has been trained upon. Under that hypothesis, both algorithms strictly operate inside the *surrounding hypercube* containing all data in D, by trying to find a partitioning of the surrounding hypercube such that, for each partition, the output value of R is similar for all samples contained into that partition. In that case, both can produce a list of if-then-else rules approximating the behaviour of R, one for each partition selected by the algorithm.

Of course, finding *fewer* relevant partitions implies producing more concise rule lists, which are more easily grasped by humans. Accordingly, GridEx differs from Iter in the way partitions are computed, and relevant hypercubes are selected. In fact, while Iter relies on a bottom-up strategy – starting from infinitely small hypercubes containing just one input space point and expanding them as much as possible –, GridEx adopts a top-down strategy—starting from a single partition containing the whole input space and recursively splitting it for a user-defined amount of times, into partitions of equal size. Thanks to this strategy, GridEx actually succeeds in finding *fewer* partitions w.r.t. Iter, while producing rules retaining a good fidelity w.r.t. R. After every split, GridEx attempts to merge couples of adjacent partitions—provided that all the samples therein contained yield similar values for R. Split and merge phases are alternated until a stopping criterion is met.

User can choose between two stopping criteria – not necessarily mutually-exclusive –, one based on the similarity (w.r.t. R) among the samples in the current hypercube, the other considering whether a maximum number of iterations has been reached or not. More precisely, if the standard deviation of the R output values of some hypercube exceeds a given threshold, then that hypercube is further partitioned: when all hypercubes are under threshold the algorithm terminates. The threshold value is a trade-off between *sensitivity* – intended as how similar should be samples grouped together – and number of rules extracted so far—i.e., the more increases the sensitivity, the more the output rules will be.

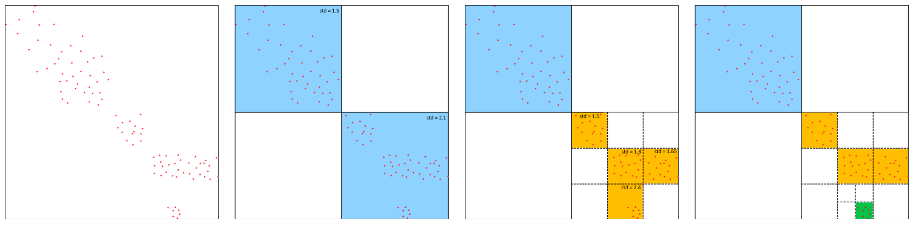
This procedure may eventually bring to the creation of *adjacent* hypercubes with similar averaged values of R. Thus, after each split and before proceeding to the successive iteration, the algorithm tries to pair-wise *merge* similar hypercubes so as to reduce the total amount of hypercubes—and thus to preserve the model interpretability. More precisely, two adjacent hypercubes are merged only if the standard deviation of R for the samples belonging to the merged hypercube does not exceed a given threshold. In this way, GridEx attains larger hypercubes without affecting the predictive performance of the resulting rules.

Overall, GridEx relies upon $n + 3$ user-defined parameters, being $n \in \mathbb{N}_{>0}$ the maximum amount of iterations it performs. Such parameters are: n, θ, m and $p_1, \ldots, p_n$, where $\theta \in \mathbb{R}_{\geq 0}$ is the similarity threshold, m is the minimum amount of samples to be considered in non-empty hypercubes, and p_i is the number of slices the algorithm performs along each dimension of the current hypercube during the i-th iteration. In the remainder of this section, we use P to denote $\langle p_1, \ldots, p_n \rangle$, $k = \dim(D)$ to denote the dimension of the input space D.

Under such hypotheses, a formal definition of GridEx is provided in Algorithm 1. Intuitively, the operation of the algorithm can be described as follows. It firstly computes the surrounding hypercube containing all data in D by finding the minimum and maximum value of each input variable. Then it recursively splits such hypercube into p_i parts along each direction, n times, therefore producing p_i^k adjacent and non-overlapping partitions of equal size, at each step. Only non-empty partitions (w.r.t. the data in D) are taken into account by the algorithm. Similarly, partitions containing samples whose standard deviation (w.r.t. R) is greater than θ are further partitioned in successive steps of the algorithm. It may happen, however, that some partition contains too few samples to provide precise predictions. To prevent this, before computing standard deviation, GridEx generates m new random samples, using R as an oracle.

Algorithm 1. GridEx pseudocode

```
Require: parameters $n, \theta, m, p_1, \dots, p_n$ to be provided
 1: function GRIDEX($R$, $D$)
 2:     $H_0 \leftarrow$ SURROUNDINGHYPERCUBE($D$)
 3:     return SLPIT(1, $H_0$, $R$, $D$)
 4: function SURROUNDINGHYPERCUBE($D$)
 5:     return the minimal hyper-cube that includes all the samples of $D$
 6: function SPLIT($i$, $H$, $R$, $D$)
 7:     if $i > n$ then return $\{H\}$
 8:     $\Pi \leftarrow \varnothing$,    $\Pi' \leftarrow \varnothing$
 9:     for all $H' \in$ PARTITIONS($H$, $p_i$) s.t. $H' \cap D \neq \varnothing$ do
10:         $D \leftarrow D \cup$ GENERATESAMPLESIN($H'$)
11:         if STDDEV($H'$, $R$, $D$) $\leq \theta$ then
12:             $\Pi \leftarrow \Pi \cup \{H'\}$
13:         else
14:             $\Pi' \leftarrow \Pi' \cup \{H'\}$
15:     $\Pi'' \leftarrow$ MERGE($\Pi, R, D$)
16:     for all $H' \in \Pi'$ do
17:         $\Pi'' \leftarrow \Pi'' \cup$ SPLIT($i + 1$, $H'$, $R$, $D$)            ▷ Recursion!
18:     return $\Pi''$
19: function GENERATESAMPLESIN($H$)
20:     return $\{m$ random points in $H\}$
21: function PARTITIONS($H$, $p$)
22:     return {all $p^k$ partitions of $H$ after splitting each edge into $p$ parts}
23: function MERGE($\Pi, R, D$)
24:     $C \leftarrow$ ADJACENTCOUPLES($\Pi$)
25:     while ($|C| > 0$) do
26:         $(H_1^*, H_2^*) \leftarrow \underset{(H_1, H_2) \in C}{\arg\min} \{$STDDEV$(H_1 \cup H_2, D, R)\}$
27:         $H \leftarrow H_1^* \cup H_2^*$
28:         if STDDEV($H$, $R$, $D$) $\leq \theta$ then
29:             $\Pi \leftarrow \Pi \setminus \{H_1^*, H_2^*\} \cup \{H\}$
30:             $C \leftarrow$ ADJACENTCOUPLES($\Pi$)
31:         else
32:             return $\Pi$
33:     return $\Pi$
34: function STDDEV($H$, $R$, $D$)
35:     return the standard deviation of all $\{R(x) \mid x \in H \cap D\}$
36: function ADJACENTCOUPLES($\Pi$)
37:     return $\{(H_1, H_2) \mid H_1, H_2 \in \Pi \land (H_1$ and $H_2$ are adjacent$)\}$
```



(a) Surrounding cube (b) Iteration 1 ($p_1 = 2$) (c) Iteration 2 ($p_2 = 2$). (d) Iteration 3 ($p_3 = 2$).

Fig. 2. Example of GridEx hyper-cube partitioning (merging step not reported). (Color figure online)

3.2 An Example

An example of hypercube partitioning executed by GridEx is reported in Fig. 2. The merging phase is not represented. The data set has two input variables (i.e. $k = 2$); user-defined parameters are $n = 3$, $P = \langle 2, 3, 2 \rangle$ and $\theta = 2.0$. In particular, Fig. 2a depicts the surrounding cube and the data set samples, represented by red dots. After first iteration (Fig. 2b), the surrounding cube is split into 4 (p_1^k) hypercubes (continuous lines), as $p_1 = 2$. The bottom-left and top-right ones are discarded as they are empty (white background). The top-left hypercube standard deviation (1.5) does not exceed θ (2.0), so it is not partitioned any further. Conversely, the fourth hypercube (standard deviation 2.1) must be further partitioned: thus, in the second iteration, it is split into 9 (p_2^k) partitions (dashed lines, Fig. 2c), as $p_2 = 3$. The same logic is then recursively applied to the 9 new hypercubes, leading to the final stage in Fig. 2d: 5 hypercubes out of 9 are discarded as empty (white background), 3 remain unaffected as their standard deviation is lower than θ (orange background), whereas the remaining one is partitioned into 2^2 smaller partitions (dotted lines), as $p_3 = 2$. Finally, of these 4 partitions, only 1 is non-empty then kept (green background).

3.3 GridEx Adaptive Splitting

Operationally, GridEx partitions a hypercube by relying on the parameters $P = \langle p_1, \dots, p_n \rangle$, which essentially state how many splits must be performed per dimension, at each step of the algorithm. In practice, the actual values of the many p_i greatly impact on the outcome of GridEx—both in terms of efficiency of the whole procedure, and in terms of predictive performance (and complexity) of the resulting partitioning. However, they still implicitly rely on the assumption that all the input variables present a comparable relevance w.r.t. the output value definition—i.e. the expected output can be more dependent or almost not dependent at all on some variables rather than others.

Accordingly, we argue that a wiser strategy in hypercubes partitioning could rely on the following insight: more *relevant* dimensions of a hypercube should be split into more parts, whereas less relevant ones can be split in fewer parts. Thus, to further optimise the amount of selected hypercubes – without sacrificing predictive performance –, users of GridEx may adopt an *adaptive* splitting. When in adaptive splitting mode, GridEx takes into account the *relevance* of each dimension of D w.r.t. the expected output value. In particular, it uses an importance measure to choose the number of splits for each dimension of a hypercube.

Importance values can be estimated in several ways. Among the many methods available – e.g. [2,44,45] – here we leverage on a simple feature selection method, SciKit-Learn's `feature_selection.f_regression`[1]. It consists of a sequential algorithm aimed at iteratively and greedly selecting the most relevant features of a dataset. It starts by training a temporary regressor on a single feature – namely, the most correlated w.r.t. the output values – and it keeps repeating this operation by adding one feature at a time, always peaking the one that mostly increases the temporary regressor predictive performance. At the end of this process, features are ranked w.r.t. their relevance, and such ranking is used for the adaptive splitting.

Accordingly, users willing to exploit adaptive splitting should specify the number of partitions assigned to each dimension on the basis of the importance calculated w.r.t. D. Importance values are normalised into range $[0, 1]$ so that the value of the most important value is 1. The algorithm should then be provided with an increasingly-monotone function of the form $f : [0, 1] \to \mathbb{N}$ to set the splits to perform. For instance, a reasonable enhancement for standard two-split iterations could be: a single split along dimensions whose importance is <0.1, two splits if the importance is between 0.1 and 0.65, and three splits otherwise.

We remark that there exists no fixed optimal choices for f: yet, a trial-and-error approach can possibly lead to quickly find the most suitable combination.

3.4 Parameter Tuning

As described in Sect. 3.1, GridEx relies on a number of parameters.

The similarity threshold θ depends on the problem under study, i.e. on the problem output value distribution, and on the trade-off between the interpretable prediction performance and the total number of extracted rules. Small values of θ lead to more rules with higher predictive performance. Conversely, large values produce less rules at the expense of the predictive performance.

As for the m parameter – representing the minimum amount of considered samples in each cube –, our experiments showed that it does not notably influence the final results. Values ranging from 10 to 100 are a good choice. We adopted $m = 15$ for the experiments reported in this paper.

For both the standard and adaptive versions of GridEx, one or two iterative partitions are enough to obtain very good results with a limited amount of output rules. Our experiments showed that larger values lead to an explosion of the rule number without any significant enhancement of the predictive performance.

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

Similarly, we found that values of 2 or 3 are suitable for the various p_i parameters identifying the number of partitions to create on each cube dimension. Larger values should be avoided, adopting instead an additional partitioning step. For instance, $P = \langle 4 \rangle$ gives the same predictive performance than $P = \langle 2, 2 \rangle$, but producing a great excess of hypercubes. This fact occurs because the single-step partitioning is equivalent to a 2-step partitioning where all the hypercubes created during the former iteration are further split during the latter.

When the output is not satisfying, it is possible to take advantage of the adaptive splitting for reducing the number of output rules with negligible deterioration of the predictive performance following the aforementioned suggestions.

4 Assessment of GridEx

In this section we provide a numerical analysis of GridEx under an explainability perspective. More precisely, our analysis is aimed at understanding if and to what extent GridEx: *(i)* is capable to approximate a black-box regressor, *(ii)* performs better than Iter in doing so, and *(iii)* is capable to provide concise and intelligible explanations for regression tasks.

Accordingly, we construct our experiments as follows. We implement both GridEx and Iter in Python and run them on a pool of black-box regressors – trained on many publicly-available data sets of growing size and dimensionality – to compare their rule-extraction capabilities. We also compare the complexity of the partitioning produced by GridEx and Iter with the ones produced by the Cart decision tree regressor trained on the same data[2]. Similarly, the data sets for our experiments are summarised in Table 1, providing, for each data set, *(i)* a bibliographic reference, *(ii)* the number of input features, *(iii)* the total number of instances, *(iv)* the percentage of samples taken apart as test set while training a black-box regressor on that data set, and *(v)* the performance of the black-box regressor. In particular, our black-box regressors are ANNs with one or two hidden layers, depending on the data set. The predicting performance of the ANN is reported in terms of MAE and R^2 value averaged on 100 tests.

Generally speaking, our experiments show how GridEx performs better than both Iter – as it produces partitionings containing *fewer* hypercubes, while attaining rule lists with better predictive performances – and Cart—as it produces simpler rule lists having choice points.

4.1 Iter Experimental Analysis

As a first step, we run Iter on all the aforementioned black-box regressors and data sets. Experiments concerning each data set are repeated 10 times. The averaged results of such experiments are summarised in Table 2.

In all the experiments, Iter parameters are the same: the number of initial hypercubes is set to 1 for all data sets, while the update parameter is chosen

[2] For the sake of *reproducibility*, the source code of our experiments is publicly available at https://github.com/sabbatinif/GridEx.

Table 1. Overview of the adopted data sets and the performances of the black-box regressors trained upon them.

Data set name	Acron.	Ref.	Features	Instances	Test set (%)	MAE	R^2
ARTI1 ($\alpha = 0$)	ARTI1	[27]	2	1 000	50	0.01	0.99
Combined Cycle Power Plant	CCPP	[17]	4	9 568	20	4.16	0.89
Airfoil Self-Noise	ASN	[1]	5	1 503	20	2.04	0.85
Energy Efficiency	EE	[20]	8	768	20	2.70	0.87
Gas Turbine CO and NOx Emission	GAS	[23]	10	36 733	20	3.16	0.84
Wine Quality	WQ	[43]	11	6 497	20	0.60	0.31

Table 2. Results of Iter applied to the data sets described in Table 1.

Data set	ARTI1	CCPP	ASN	EE	GAS	WQ
# features	2	4	5	8	10	11
Threshold	0.2	7.0	4.0	4.0	15.0	2.0
# iterations	26	600	582	600	600	600
# hyper-cubes	4	329	113	55	44	23
# useful hyper-cubes	4	16	64	25	14	11
Coverage (%)	100.0	91.8	97.9	83.5	76.6	62.3
Left training samples (%)	0.0	8.4	0.0	2.9	6.6	6.1
Missed test samples (%)	0.0	9.1	1.7	3.3	6.7	7.8
MAE (data)	0.04	5.37	4.24	3.52	8.77	0.73
MAE (ANN)	0.04	3.77	3.40	2.37	7.88	0.55
R^2 (data)	0.92	0.83	0.40	0.76	0.10	−0.05
R^2 (ANN)	0.92	0.91	0.55	0.92	0.17	−0.12

as double w.r.t. the predefined one described by the authors of Iter—i.e. 0.1 instead of 0.05. This aims at reducing the amount of iterations required by Iter to converge, especially with the more complex data sets, provided that the algorithm terminates when either 600 iterations are performed, or *all* the training samples have been covered by the created hypercubes.

Accordingly, for each data set we collect *(i)* the overall number of actually useful hypercubes – i.e. those containing at least one training sample – found by Iter, *(ii)* the total number of iterations performed by the algorithm, *(iii)* the selected threshold parameter value, *(iv)* the amount of input space covered by the hypercubes expressed in percentage, *(v)* the percentage of training samples that are not included in any hypercube and, analogously, *(vi)* the percentage of test samples that the output model is not able to predict. To improve readability, the feature number of each data set is reported in Table 2 as well. Finally, the MAE and R^2 scores of all Iter predictions are reported w.r.t. both the original data set and the black-box predictions. These latter measurements are performed using the test-set samples—which in turn are *never* used for training.

Table 2 highlights that in 4 cases out of 6 the algorithm reaches the maximum allowed iterations without covering the whole training set. When this is the case, the resulting rule list produced by Iter is affected by a reduced predictive capability, w.r.t. the black-box regressor it mimics. This effect can be detected by comparing their test-set performance. Such non-exhaustivity issue is particularly evident for the ASN data set as well.

Generally speaking, the overall Iter performance is very good with simple data sets (e.g., the ARTI1 data set). However, we observe a degradation as the complexity of the data set grows. In any case, both performance and computational cost heavily depend on the parameter values and initial conditions, such as the starting cube number and position. Parameter tuning can be performed through a trial-and-error approach [27], but this often implies a trade-off between execution time, number of extracted rules and result accuracy.

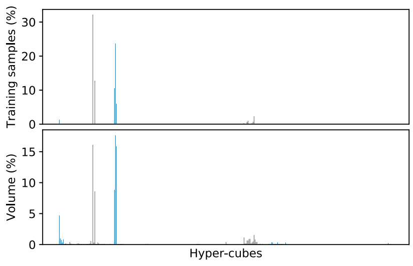
To better analyse how Iter attempts to address the non-exhaustivity issue, the left side of Fig. 3 depicts several plots describing executions of the algorithm on different data sets and black boxes. Each plot has two panels. In both panels the horizontal axis refers to the computational time (from left to right), whereas the vertical bars refer to hypercubes. So left-most bars refer to hypercubes which are found *earlier*. Top panels represent the number of samples belonging to each hypercube, expressed as the percentage of the overall training examples. Conversely, bottom panels represent the relative volume of each hypercube, expressed as the percentage of the whole input feature space.

Notably, we observe that later iterations of Iter tend to create smaller hypercubes that include less samples than the ones computed in previous iterations (cf. the GAS and WQ data sets). Exceptions may occur when samples are not uniformly distributed within the input space. This happens for instance in the EE data set, where it is possible to find big hypercubes with few samples and, conversely, very small cubes including up to a fourth of the training set. Finally, the CCPP data set is a perfect example of Iter uncontrolled hypercube expansion towards *irrelevant* input space regions: more than 95% of the hypercubes have no predictive relevance. The algorithm wastes time and resources exploring these regions, reaching the maximum iteration number with almost a 10% of the samples uncovered by the interpretable model rules.

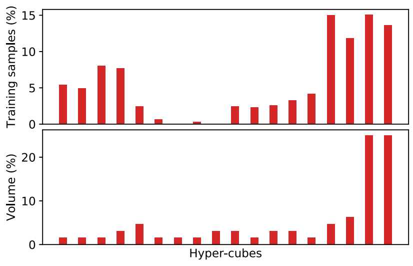
As discussed below, GridEx overcomes those drawbacks by achieving better predictive performance in less time and with a lower computational effort.

4.2 GridEx Experimental Analysis

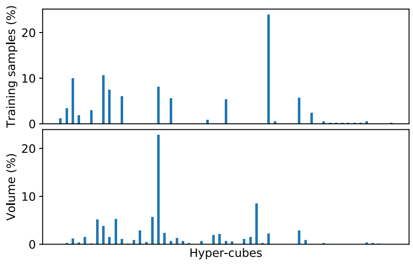
To fairly compare Iter and GridEx, we evaluate the latter as well against all the aforementioned black-box regressors and data sets. Results are reported in Table 3, where rows and columns retain the same meaning as in Table 2, except for the "Left training samples" row, missing. Indeed, it is not useful to report the number of training samples left out by the algorithm, as GridEx always covers the entire training set, by construction. Furthermore, the user-defined partitions



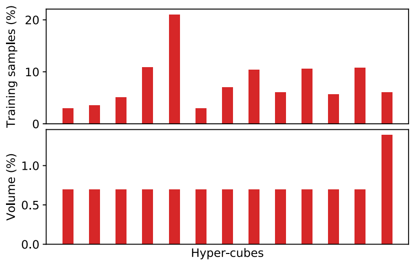
(a) Iter on CCPP.



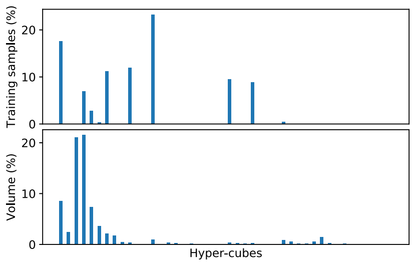
(b) GridEx on CCPP.



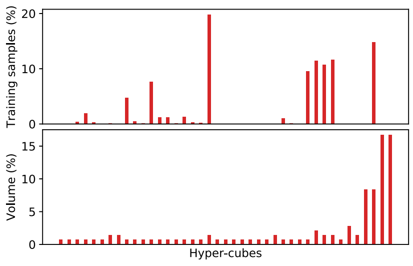
(c) Iter on EE.



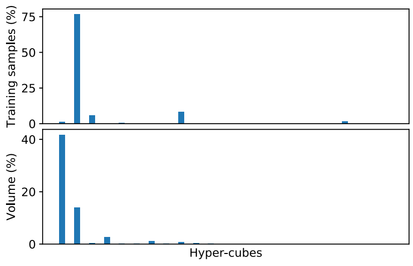
(d) GridEx on EE.



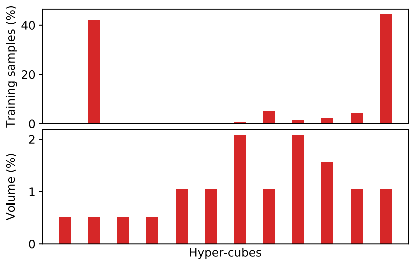
(e) Iter on GAS.



(f) GridEx on GAS.



(g) Iter on WQ.



(h) GridEx on WQ.

Fig. 3. Number of training examples (top panels) and volume (bottom panels) of each hyper-cube created by Iter (left plots) and GridEx (right plots). Values are expressed as percentage of the total number of training samples and the surrounding cube volume, respectively. Each row represents a different data set.

Table 3. Results of GridEx applied to the data sets described in Table 1.

Data set	ARTI1	CCPP	ASN	EE	GAS	WQ
# features	2	4	5	8	10	11
Threshold	0.01	4.10	4.00	2.00	7.00	1.00
Partitions	⟨2⟩	⟨a, a⟩	⟨a, a⟩	⟨a⟩	⟨a, a⟩	⟨a⟩
Feature importance, adaptive partitions	–	≤0.04, 1 ≤0.5, 2 ≤1, 4	≤0.2, 1 ≤0.5, 2 ≤0.7, 3 ≤1, 4	≤0.001, 1 ≤0.5, 2 ≤1, 3	≤0.1, 1 ≤0.7, 3 ≤1, 4	≤0.03, 1 ≤0.6, 2 ≤1, 3
# hyper-cubes	4	18	38	13	41	12
Coverage (%)	100.0	93.8	43.2	9.7	84.0	13.0
Missed test samples (%)	0.0	0.0	1.3	0.0	0.0	0.1
MAE (data)	0.01	4.37	3.27	2.65	6.79	0.69
MAE (ANN)	0.01	2.61	2.57	1.23	5.51	0.38
R^2 (data)	1.00	0.89	0.66	0.88	0.48	0.17
R^2 (ANN)	0.99	0.96	0.76	0.98	0.61	0.46

are reported as either integer numbers for the standard GridEx or as ‘a’ for the adaptive variant. When the adaptive option is chosen, the user-defined feature importance ranges and the corresponding partition numbers are also reported.

Unlike Iter, GridEx can predict almost every sample of the test set. Furthermore, the extracted rules only describe *relevant* regions of the input space. This design choice may lead to a partial coverage of the input space—e.g. for the EE and WQ data sets. Little coverage commonly occurs when the training samples are not uniformly distributed in the input space, but they are, conversely, concentrated only in some sub-regions that GridEx is able to discern and mark as relevant. However, in some corner cases, GridEx may require a larger number of hypercubes to achieve a *significantly* better predictive performance than Iter: then, it produces more rules at the expense of readability of the final rule set.

The right side of Fig. 3 depicts plots describing executions of GridEx on different data sets and black boxes, as previously described for Iter. The plots show how, in general, GridEx produces fewer hypercubes than Iter, proving itself more concise on most data sets. There are, however, notable exceptions—such as the GAS data set, where Iter produces slightly fewer hypercubes than GridEx (considering only the relevant ones). So, also GridEx may sometimes find hypercubes with little predictive relevance, for the shortage of examples contained: in that case, a pruning algorithm could be exploited to reduce the number of extracted rules with no relevant impact on the overall performance.

Also, some peculiar features of GridEx are effective in overcoming Iter, in the general case. For instance, the iterative multi-level partitioning performed by GridEx is a very flexible feature, enabling users to tune the refinement of those inaccurate rules that include samples with larger standard deviations. However, to be effective, it must be carefully tuned, as the number of hypercubes may grow

Table 4. Results of CART applied to the data sets in Table 1.

Data set	ARTI1	CCPP	ASN	EE	GAS	WQ
# leaves	4	15	50	40	50	15
MAE (data)	0.01	4.49	3.06	2.78	4.88	0.65
MAE (ANN)	0.01	2.44	2.08	0.59	3.53	0.24
R^2 (data)	1.00	0.88	0.70	0.86	0.67	0.25
R^2 (ANN)	0.99	0.96	0.80	0.99	0.78	0.71

exponentially when samples are uniformly distributed among the entire input space. Similarly, GridEx merging phase supports the creation of compact rule lists with less, coarser-grained rules and no predictive performance degradation.

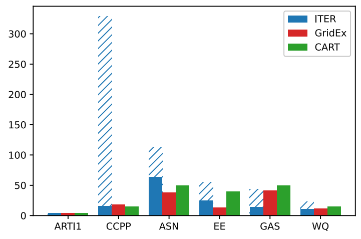
4.3 Comparison of ITER and GridEx

Finally, to perform an unbiased comparison between GridEx and ITER, a decision tree regressor [11] is taken as a reference—similarly to [12]. More precisely, we adopt the CART algorithm for each data set in Table 1, with a maximum depth parameter equal to twice the amount of input features. The resulting decision trees are summarised in Table 4, in terms of MAE and R^2 value w.r.t. both the data and the underlying neural network. The number of leaves is reported as well since it determines the number of decision rules exploited by the regressor.

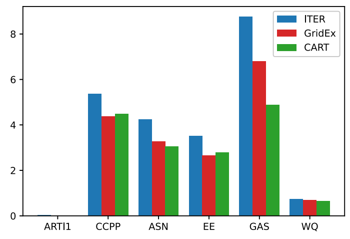
The choice of the maximum depth parameter comes from the following considerations. ITER and GridEx rules are expressed as hypercubes. Every cube has 2 constraints on each input variable, i.e. the lower and upper values. Since each node of the decision tree represents a constraint on a variable, our choice of the maximum depth parameter ensures that the resulting decision rules cannot be subject to more constraints than the rules produced by either ITER or GridEx.

Figure 4 summarises results of the comparison among the three extraction procedures. GridEx (Fig. 4a) produces an equal or smaller number of rules w.r.t. both ITER and CART in almost all the cases—even if we cut off the irrelevant hypercubes produced by ITER. Since interpretability of a rule list decreases with its length, we argue that GridEx produces more interpretable results.

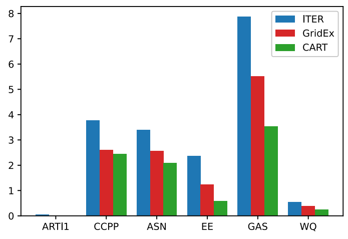
As for the predictive performance, GridEx is always better than ITER in terms of MAE (Figs. 4b and 4c) and R^2 value (Figs. 4d and 4e), with respect to both the data and the underlying black-box predictions. However, CART has generally smaller MAE and larger R^2 than GridEx. This can be explained by the undesired discretisation introduced by the constant output value of GridEx and ITER rules, as well as by the higher amount of rules extracted by CART.



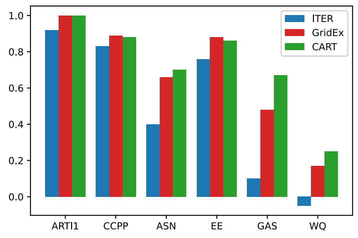
(a) Number of extracted rules.



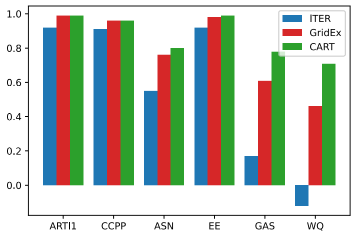
(b) MAE with respect to the data.



(c) MAE fidelity with respect to the underlying black box.



(d) R^2 value with respect to the data.



(e) R^2 value with respect to the underlying black box.

Fig. 4. Comparison between ITER, GridEx, and CART using both MAE (the lower the better) and R^2 scores (the higher the better).

5 Conclusions

In this paper we present GridEx, a new pedagogical knowledge-extraction procedure aimed at globally explaining black-box regressors. GridEx extends ITER by overcoming some of its major limitations—namely, its non-exhaustivity and its tendency to focus on non-interesting regions of the input space. GridEx is able to create an interpretable approximation of any black-box regressor in the form of decision rules. W.r.t. ITER, GridEx produces fewer decision rules, while attaining better predictive performance, even when the data set is high-dimensional.

In our next research efforts we plan to extend GridEx so as to address other limitations of Iter, such as the inability to handle categorical input features – with no pre-processing – and the constant output value of its decision rules. Furthermore, we intend to design an automatic procedure regarding the best adaptive splitting parameter selection. We also plan to extend our comparative numerical analysis to the other rule-extraction algorithms for regression mentioned in Sect. 2.1.

Acknowledgements. This paper has been partially supported by *(i)* the H2020 project "StairwAI" (G.A. 101017142), and *(ii)* the CHIST-ERA IV project "EXPECTATION" (G.A. CHIST-ERA-19-XAI-005).

References

1. Airfoil Self-Noise Data Set (2014). https://archive.ics.uci.edu/ml/datasets/Airfoil+Self-Noise. Accessed 19 Jan 2021
2. Altmann, A., Toloşi, L., Sander, O., Lengauer, T.: Permutation importance: a corrected feature importance measure. Bioinformatics **26**(10), 1340–1347 (2010)
3. Andrews, R., Diederich, J., Tickle, A.B.: Survey and critique of techniques for extracting rules from trained artificial neural networks. Knowl. Based Syst. **8**(6), 373–389 (1995)
4. Ayache, S., Eyraud, R., Goudian, N.: Explaining black boxes on sequential data using weighted automata. In: International Conference on Grammatical Inference, pp. 81–103. PMLR (2019)
5. Azcarraga, A., Liu, M.D., Setiono, R.: Keyword extraction using backpropagation neural networks and rule extraction. In: The 2012 International Joint Conference on Neural Networks (IJCNN), pp. 1–7. IEEE (2012)
6. Baesens, B., Setiono, R., De Lille, V., Viaene, S., Vanthienen, J.: Building credit-risk evaluation expert systems using neural network rule extraction and decision tables. In: ICIS 2001 Proceedings, vol. 20 (2001). http://aisel.aisnet.org/icis2001/20
7. Baesens, B., Setiono, R., Mues, C., Vanthienen, J.: Using neural network rule extraction and decision tables for credit-risk evaluation. Manage. Sci. **49**(3), 312–329 (2003)
8. Barakat, N., Diederich, J.: Eclectic rule-extraction from support vector machines. Int. J. Comput. Intell. Appl. **2**(1), 59–62 (2005)
9. Bologna, G.: A study on rule extraction from neural networks applied to medical databases. In: The 4th European Conference on Principles and Practice of Knowledge Discovery (PKDD2000) (2000)
10. Bologna, G., Pellegrini, C.: Three medical examples in neural network rule extraction. Physica Medica **13**, 183–187 (1997)
11. Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A.: Classification and Regression Trees. CRC Press (1984)
12. Calegari, R., Ciatto, G., Dellaluce, J., Omicini, A.: Interpretable narrative explanation for ML predictors with LP: a case study for XAI. In: Bergenti, F., Monica, S. (eds.) WOA 2019–20th Workshop "From Objects to Agents", CEUR Workshop Proceedings, vol. 2404, pp. 105–112. Sun SITE Central Europe, RWTH Aachen University, 26–28 June 2019. http://ceur-ws.org/Vol-2404/paper16.pdf

13. Calegari, R., Ciatto, G., Omicini, A.: On the integration of symbolic and sub-symbolic techniques for XAI: a survey. Intelligenza Artificiale **14**(1), 7–32 (2020). https://doi.org/10.3233/IA-190036
14. Ciatto, G., Calegari, R., Omicini, A., Calvaresi, D.: Towards XMAS: eXplainability through Multi-Agent Systems. In: Savaglio, C., Fortino, G., Ciatto, G., Omicini, A. (eds.) AI&IoT 2019 - Artificial Intelligence and Internet of Things 2019, CEUR Workshop Proceedings, vol. 2502, pp. 40–53. Sun SITE Central Europe, RWTH Aachen University, November 2019
15. Ciatto, G., Calvaresi, D., Schumacher, M.I., Omicini, A.: An abstract framework for agent-based explanations in AI. In: El Fallah Seghrouchni, A., Sukthankar, G., An, B., Yorke-Smith, N. (eds.) 19th International Conference on Autonomous Agents and MultiAgent Systems, pp. 1816–1818. IFAAMAS, May 2020. http://ifaamas.org/Proceedings/aamas2020/pdfs/p1816.pdf
16. Ciatto, G., Schumacher, M.I., Omicini, A., Calvaresi, D.: Agent-based explanations in AI: towards an abstract framework. In: Calvaresi, D., Najjar, A., Winikoff, M., Främling, K. (eds.) EXTRAAMAS 2020. LNCS (LNAI), vol. 12175, pp. 3–20. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51924-7_1
17. Combined Cycle Power Plant Data Set (2014). https://archive.ics.uci.edu/ml/datasets/Combined+Cycle+Power+Plant. Accessed 19 Jan 2021
18. Craven, M.W., Shavlik, J.W.: Using sampling and queries to extract rules from trained neural networks. In: Machine Learning Proceedings 1994, pp. 37–45. Elsevier (1994). https://doi.org/10.1016/B978-1-55860-335-6.50013-1
19. Craven, M.W., Shavlik, J.W.: Extracting tree-structured representations of trained networks. In: Touretzky, D.S., Mozer, M.C., Hasselmo, M.E. (eds.) Advances in Neural Information Processing Systems 8. Proceedings of the 1995 Conference, pp. 24–30. The MIT Press, June 1996
20. Energy Efficiency Data Set (2012). https://archive.ics.uci.edu/ml/datasets/Energy+efficiency. Accessed 19 Jan 2021
21. Franco, L., Subirats, J.L., Molina, I., Alba, E., Jerez, J.M.: Early breast cancer prognosis prediction and rule extraction using a new constructive neural network algorithm. In: Sandoval, F., Prieto, A., Cabestany, J., Graña, M. (eds.) IWANN 2007. LNCS, vol. 4507, pp. 1004–1011. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73007-1_121
22. Freitas, A.A.: Comprehensible classification models: a position paper. ACM SIGKDD Explor. Newslett. **15**(1), 1–10 (2014)
23. Gas Turbine CO and NOx Emission Data Set (2019). https://archive.ics.uci.edu/ml/datasets/Gas+Turbine+CO+and+NOx+Emission+Data+Set. Accessed 19 Jan 2021
24. Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., Pedreschi, D.: A survey of methods for explaining black box models. ACM Comput. Surv. (CSUR) **51**(5), 1–42 (2018)
25. Hayashi, Y., Setiono, R., Yoshida, K.: A comparison between two neural network rule extraction techniques for the diagnosis of hepatobiliary disorders. Artif. Intell. Med. **20**(3), 205–216 (2000)
26. Hofmann, A., Schmitz, C., Sick, B.: Rule extraction from neural networks for intrusion detection in computer networks. In: SMC 2003 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme-System Security and Assurance (Cat. No. 03CH37483), vol. 2, pp. 1259–1265. IEEE (2003)

27. Huysmans, J., Baesens, B., Vanthienen, J.: ITER: an algorithm for predictive regression rule extraction. In: Tjoa, A.M., Trujillo, J. (eds.) DaWaK 2006. LNCS, vol. 4081, pp. 270–279. Springer, Heidelberg (2006). https://doi.org/10.1007/11823728_26
28. Huysmans, J., Dejaeger, K., Mues, C., Vanthienen, J., Baesens, B.: An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. Decis. Support Syst. **51**(1), 141–154 (2011)
29. Johansson, U., König, R., Niklasson, L.: Rule extraction from trained neural networks using genetic programming. In: 13th International Conference on Artificial Neural Networks, pp. 13–16 (2003)
30. Lipton, Z.C.: The mythos of model interpretability. Queue **16**(3), 31–57 (2018). https://doi.org/10.1145/3236386.3241340
31. Martens, D., Baesens, B., Van Gestel, T., Vanthienen, J.: Comprehensible credit scoring models using rule extraction from support vector machines. Eur. J. Oper. Res. **183**(3), 1466–1476 (2007)
32. Murphy, P.M., Pazzani, M.J.: Id2-of-3: Constructive induction of m-of-n concepts for discriminators in decision trees. In: Machine Learning Proceedings 1991, pp. 183–187. Elsevier (1991)
33. Quinlan, J.R.: Simplifying decision trees. Int. J. Man Mach. Stud. **27**(3), 221–234 (1987). https://doi.org/10.1016/S0020-7373(87)80053-6
34. Quinlan, J.R.: C4.5: Programming for Machine Learning. Morgan Kauffmann 38, 48 (1993)
35. Rocha, A., Papa, J.P., Meira, L.A.: How far do we get using machine learning black-boxes? Int. J. Pattern Recogn. Artif. Intell. **26**(02), 1261001-(1–23) (2012). https://doi.org/10.1142/S0218001412610010
36. Rudin, C.: Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nat. Mach. Intell. **1**(5), 206–215 (2019). https://doi.org/10.1038/s42256-019-0048-x
37. Saito, K., Nakano, R.: Extracting regression rules from neural networks. Neural Netw. **15**(10), 1279–1288 (2002). https://doi.org/10.1016/S0893-6080(02)00089-8
38. Schmitz, G.P., Aldrich, C., Gouws, F.S.: ANN-DT: an algorithm for extraction of decision trees from artificial neural networks. IEEE Trans. Neural Networks **10**(6), 1392–1401 (1999). https://doi.org/10.1109/72.809084
39. Setiono, R., Baesens, B., Mues, C.: Rule extraction from minimal neural networks for credit card screening. Int. J. Neural Syst. **21**(04), 265–276 (2011). https://doi.org/10.1142/S0129065711002821
40. Setiono, R., Leow, W.K., Zurada, J.M.: Extraction of rules from artificial neural networks for nonlinear regression. IEEE Trans. Neural Networks **13**(3), 564–577 (2002). https://doi.org/10.1109/TNN.2002.1000125
41. Steiner, M.T.A., Neto, P.J.S., Soma, N.Y., Shimizu, T., Nievola, J.: Using neural network rule extraction for credit-risk evaluation. Int. J. Comput. Sci. Network Secur. **6**(5), 6–16 (2006)
42. Towell, G.G., Shavlik, J.W.: Extracting refined rules from knowledge-based neural networks. Mach. Learn. **13**(1), 71–101 (1993). https://doi.org/10.1007/BF00993103
43. Wine Quality Data Set (2009). https://archive.ics.uci.edu/ml/datasets/Wine+Quality. Accessed 19 Jan 2021

44. Zhuang, J., Dvornek, N.C., Li, X., Yang, J., Duncan, J.: Decision explanation and feature importance for invertible networks. In: 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), pp. 4235–4239. IEEE (2019)
45. Zien, A., Krämer, N., Sonnenburg, S., Rätsch, G.: The feature importance ranking measure. In: Buntine, W., Grobelnik, M., Mladenić, D., Shawe-Taylor, J. (eds.) ECML PKDD 2009. LNCS (LNAI), vol. 5782, pp. 694–709. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04174-7_45