Hypercube-Based Methods for Symbolic Knowledge Extraction: Towards a Unified Model

Federico Sabbatini[1,*], Giovanni Ciatto[2], Roberta Calegari[3] and Andrea Omicini[2]

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

Abstract
Symbolic knowledge-extraction (SKE) algorithms proposed by the XAI community to obtain human-intelligible explanations for opaque machine learning predictors are currently being studied and developed with growing interest, also in order to achieve believability in interactions. However, choosing the most adequate extraction procedure amongst the many existing in the literature is becoming more and more challenging, as the amount of available methods increases. In fact, most of the proposed algorithms come with *constraints* over their applicability.

In this paper we focus upon a quite general class of SKE techniques, namely *hypercube-based* methods. Despite being commonly considered as regression-specific, we discuss why hypercube-based SKE methods are flexible enough to deal with classification problems as well. More generally, we propose a common generalised model for hypercube-based methods, and we show how they can be exploited to perform SKE on datasets, predictors, or learning tasks of any sort.

Keywords
explainable AI, knowledge extraction, interpretable prediction, PSyKE

1. Introduction

One of the main features to be underpinned by a believable human-agent interaction is the capability of being *explainable*. Along this line, *symbolic knowledge extraction* (SKE) is a powerful tool within the scope of explainable artificial intelligence (XAI). It enables reverse-engineering of the black-box (BB) machine learning algorithms—which are nowadays exploited in many AI tasks [1]. SKE allows data scientists to associate human-comprehensible, *post-hoc* explanations [2] to the recommendations or decisions computed by the most common prediction-effective – yet, poorly interpretable – algorithms. To cite some examples, SKE is widely adopted to credit-risk evaluation [3, 4, 5], medical diagnosis – i.e., to make early breast cancer prognosis predictions

WOA 2022: 23rd Workshop From Objects to Agents, September 1–2, Genova, Italy
*Corresponding author.
f.sabbatini1@campus.uniurb.it (F. Sabbatini); giovanni.ciatto@unibo.it (G. Ciatto); roberta.calegari@unibo.it (R. Calegari); andrea.omicini@unibo.it (A. Omicini)
http://federicosabbatini.apice.unibo.it (F. Sabbatini); https://about.me/gciatto (G. Ciatto); http://robertacalegari.apice.unibo.it (R. Calegari); http://andreaomicini.apice.unibo.it (A. Omicini)
0000-0002-0532-6777 (F. Sabbatini); 0000-0002-1841-8996 (G. Ciatto); 0000-0003-3794-2942 (R. Calegari); 0000-0002-6655-3869 (A. Omicini)

[6] and for recognising hepatobiliary disorders [7] or other diseases and dysfunctions [8] –, credit card screening [9], intrusion detection systems [10], keyword extraction [11], and space mission data prediction [12].

The basic idea behind SKE is to construct *symbolic* (hence, interpretable) models mimicking the behaviour of the pre-existing black-box predictors to be explained. Such symbolic models should describe the corresponding black boxes in terms of the outputs they provide as responses to (classes of) inputs values. Symbols, in particular, may consist of intelligible knowledge, e.g., lists or trees of *logic rules* that can be exploited to obtain predictions as well as to better understand the underlying predictor. In other words, symbols are both human- and machine-interpretable.

Because of the many SKE techniques available in the literature, selecting the most appropriate SKE algorithm for any given learning task may easily become cumbersome. Difficulties may arise because of the intrinsic design choices behind each extraction algorithm. In fact, SKE algorithms may commonly target specific learning tasks (classification or regression), specific sorts of ML predictors (e.g. neural networks, support-vector machines, linear models, etc.), or specific sorts of training data (e.g. continuous, categorical, or binary).

In this paper we focus upon a quite general class of SKE techniques, namely *hypercube-based* methods. Methods of such sort extract symbolic knowledge by querying black-box predictors as oracles, and by recursively partitioning the input spaces of these black boxes into several hypercubes—hence following a *divide-et-impera* approach. Despite being commonly considered regression-specific, we show that hypercube-based SKE methods are flexible enough to deal with classification problems as well. More generally, we propose a common model for hypercube-based methods, and we show how they can be exploited to perform SKE on data sets, predictors, or learning tasks of any sort.

Accordingly, the contribution of this paper is manifold. First, we provide a general and abstract description of any hypercube-based SKE workflow. Second, we compare existing hypercube-based methods (e.g. Iter [13] and GridEx [14]) based on the way they partition the input space, approximate black-box decisions, and construct the extracted symbolic knowledge. Third, we discuss how hypercube-based methods can fully support supervised learning tasks—there including regression and classification ones.

The remainder of this paper is organised as follows. Section 2 describes the state of the art for SKE as well as some background notions to fully understand the work. Section 3 presents our model for hypercube-based methods and compares the most relevant methods from the literature. Finally, conclusions are drawn in Section 4.

2. State of the Art

In this section we provide readers with some details about the state of the art for symbolic knowledge extraction.

2.1. Knowledge Extraction

Computational systems are considered *interpretable* when humans are able to easily understand its operation and outcomes [15]. However, nowadays decision support systems often rely on

ML models having excellent predictive capabilities at the expense of interpretability. These *sub-symbolic* predictors of growing complexity, which learn input-output relations from data and store them as internal parameters, do not provide any kind of *symbolic* representation of the acquired knowledge, thus lacking of an interpretable representation to the benefit of human users. ML algorithms are defined as *black boxes* for this reason [16].

It is possible to preserve the impressive BB predictive performance and, at the same time, obtain human-intelligible clues or explanations regarding the BB behaviour by substituting the opaque model with a mimicking interpretable surrogate. The XAI community, indeed, have proposed a number of means to produce *ex-post* explanations for sub-symbolic predictors in the form of surrogate models based on sets of rules extracted from the underlying opaque model. Amongst the proposals there are methods to extract lists [17, 13, 14] or trees [18, 19] of logic rules, usually if-then-else, *M-of-N* or fuzzy. SKE is particularly important also for another reason: it may enable further manipulations, for instance to merge the *know-how* of different BB models [20].

Knowledge extraction algorithms can be categorised along three orthogonal dimensions [21]: *(i)* supported learning tasks, *(ii)* shape of the symbolic knowledge provided in output, *(iii)* the supported underlying ML models, usually known as *translucency*.

Supported tasks – item *(i)* – are usually supervised classification or regression. A cluster of SKE algorithms can only explain BB classifiers – e.g. Rule-extraction-as-learning [17], Trepan [18] and others [22, 23] –, while a different cluster is designed to support BB regressors—e.g., Iter [13], GridEx [14], GridREx [24] and others [25, 26, 27]. Finally, a little subset of SKE techniques are able to handle both tasks, as for the case of G-Rex [28] and Cart [19].

As for the shape of the output knowledge – item *(ii)* –, decision rules [29, 30, 31] and trees [32, 33] are usually considered the most human-understandable ways to represent knowledge. For this reason the majority of SKE methods produce one of these two structures as output. Regardless of the shape, conditions describing decision rules and nodes are expressed by using the same input/output data types adopted to train the underlying BB. For instance, SKE procedures applied to classifiers accepting N-dimensional numerical data and providing K distinct output classes will produce rule lists or trees involving a certain number of *predicates* over N input variables $x_1, \ldots, x_n$ and having K possible outcomes. A further categorisation may be performed w.r.t. the kind of predicates contained in the output knowledge. In particular, it is possible to observe conjunctions or disjunctions of inequalities (e.g. $x_i \gtrless c$) as well as inclusions in or exclusions from intervals (e.g. $x_i \in [l, u]$) for numerical data. Categorical data are usually associated to equalities (e.g. $x_i = c$) and set-inclusions (e.g. $x_i \in \{c_1, c_2, \ldots\}$). *M-of-N* or fuzzy rules are other available alternatives.

Finally, the translucency dimension – item *(iii)* – represents the strategy adopted by the SKE algorithm to obtain interpretable knowledge from a BB. In particular, extractors may be *decompositional* or *pedagogical* [34, 21]. Decompositional techniques consider the internal structure of the underlying black box, hence producing symbolic knowledge which mimics how it internally works. As a side-effect, decompositional algorithms are bound to specific sorts of ML predictors—and possibly introduce constraints on their internal structures. For instance, techniques tailored on neural networks are not applicable to support-vector machines. Similarly, procedures explicitly designed for 3-layered networks are not suitable for deeper ones. On the other hand, pedagogical methods can extract symbolic knowledge without relying on any information about the inner structure of predictors. They simply query the predictor as an oracle, observing its response to particular inputs, and generalise its behaviour accordingly. For this reason, pedagogical extractors come with no constraints on sorts of predictors they can be applied to. Hence, they are more general – despite potentially less precise –, but considerations about the output performance strictly depend on the task at hand.

To evaluate the quality of SKE techniques different indicators are exploited, depending on the task to solve. Common choices are readability, fidelity and predictive performance measurements [35]. The former expresses how interpretable is the output knowledge from the human perspective. It is generally evaluated through the number of extracted rules and the number of constraints per rule. Fidelity is related to the capability of the extracted knowledge to mimic the underlying BB predictions, whereas predictive performance measurements are assessed by comparing the predictions drawn from the extracted knowledge with the expected data. Measurements involving predictions should be assessed via the same scoring function used for the underlying BB—which in turn strictly depends on the performed task. Classifiers are usually evaluated via accuracy, precision, recall, and F_1 score. Conversely, common metrics for regressors are the mean absolute/squared error (MAE/MSE) and the R^2 score.

3. Hypercube-Based Knowledge Extractors

In this section we present a general model for hypercube-based extractors (Subsection 3.1). We then delve into the details of different algorithms from the literature, discussing how they match the model (Subsection 3.2).

3.1. Unified Model

Hypercube-based extraction methods are *pedagogical* extraction procedures which can operate on trained ML predictors of any sort. They consider the predictor P undergoing extraction as an oracle to be queried multiple times, in order to find a partitioning $H_1 \cup \ldots \cup H_n$ of its input space $\mathcal{X}$ such that the output space $\mathcal{Y}$ can be concisely expressed for each partition. Hence, they extract knowledge in the form of rule lists or trees, where each rule attempts to describe the outcome of P for a particular hypercube $H_i \subseteq \mathcal{X}$. Rules have the following logical form:

$$\mathbf{x} \in H_i \rightarrow (\ \mathbf{y} = f_i(\mathbf{x})\ )$$

to be read as "if the input vector $\mathbf{x} \in \mathcal{X}$ is in some hypercube H_i, then the prediction $\mathbf{y} \in \mathcal{Y}$ is $f_i(\mathbf{x})$", where $f_i(\mathbf{x})$ is a function that approximates P outcomes. Note that each hypercube H_i is a partition of the input space $\mathcal{X}$, and f_i is the function approximating P outcomes related to that hypercube (could be a costant, a linear function, etc.), i.e.: $f_i(\mathbf{x}) \approx P(\mathbf{x}),\ \forall \mathbf{x} \in H_i$.

Hypercube-based extraction procedures should then attempt to select hypercubes and local approximation functions so as to maximise the fidelity of the overall rule set/list w.r.t. P. Accordingly, they follow a pretty linear workflow, which may be roughly summarised in 3 steps, namely:

1. partitioning the input space into disjoint hypercubes $H_1, \ldots, H_n$, following a selected strategy and according to possible defined constraints;

    - e.g. one may be willing to minimise n while maximising the size of each H_i

2. approximating the prediction of P for each hypercube H_i, via some function f_i;
    - e.g. one may be willing to maximise the similarity among P and f_i in H_i

3. creating a rule set where each rule *concisely* represents the behaviour of P in H_i via f_i.

In the following, we delve into the details of all the aforementioned phases.

3.1.1. Input space partitioning

Input space partitioning is a recursive computation aimed at finding the optimal number, shape and size of hypercubes w.r.t. some *desiderata*, such as: *(i)* covering the whole input space; *(ii)* obtaining disjoint regions; *(iii)* minimising the number of regions; *(iv)* maximising the similarity amongst the samples inside single regions; *(v)* minimising the predictive error correlated to each partition.

These conditions cannot be all satisfied simultaneously, especially when dealing with high-dimensional data sets. Thus, some requirements may be relaxed. For instance, the input space coverage may be limited only to *interesting* hypercubes, neglecting the others. Let us consider a hypercube 'interesting' if it contains training samples, as it may have a role to play in drawing future predictions.

Alternatively, the partitioning process may terminate after a predefined number of iterations, as some state-of-the-art algorithms actually do. However, this may lead to the indiscriminate exclusion of some regions of the input space that, conversely, are not negligible. In turn, the explained model will not be able to provide predictions for a subset of input instances.

Non-contiguous hypercubes may be relaxed into hierarchical or fuzzy regions, possibly mapped into non-overlapping rules, in order to have unambiguous output predictions.

Similarity and fidelity The amount of hypercubes an input space is partitioned into may significantly impact the interpretability of the final symbolic model. In fact, hypercube-based methods will output as many rules as the hypercubes they have partitioned the input space into—and of course more (or more complex) rules imply lower readability for the human user.

The capability of grouping together similar samples into a single hypercube is so quintessential for supporting the creation of few, general, and simple rules which capture the behaviour of the original predictor with high fidelity. This corresponds to *(i)* data points from the same hypercube drawing *similar* predictions, and to *(ii)* predictions having a high *fidelity* (or, equivalently, low error rates) w.r.t. to the original predictor. Accordingly, here we delve into the details of how to assess *(i)* similarity amongst data points from *contiguous* hypercubes, as well as *(ii)* predictive errors between a candidate rule and the underlying predictor.

Similarity amongst instances Input space partitions may be considered similar according to the following definitions:

input closeness if the input variables of both subregions have values ranging in similar domains, as for the case of adjacent disjoint or overlapping regions;

output closeness if the output associated with the instances in the two subregions may be defined as similar.

While it is straightforward to check input closeness (e.g., through Euclidean distance), dealing with output closeness requires taking into account the learning problem at hand.

As far as classification is concerned, we may consider two hypercubes H_1 and H_2 as *output-close* w.r.t. a predictor P if (and only if) the most frequent output class is the same in both hypercubes:

$$H_1 \overset{P}{\approx} H_2 \Leftrightarrow \text{mode}(P(H_1)) = \text{mode}(P(H_2)) \tag{1}$$

where mode$(\cdot)$ denotes the statistical operator returning the most frequent item over a set, and $P(H)$ is a shortcut standing for $\{P(\mathbf{x}) : \mathbf{x} \in H\}$, to lighten the notation.

Conversely, in the case of regression with constant outputs, output-similarity may be expressed as a function of the absolute difference between the mean output predictions performed by the predictor P on the two hypercubes H_1 and H_2:

$$H_1 \overset{P}{\approx} H_2 \Leftrightarrow |\text{mean}(P(H_1)) - \text{mean}(P(H_2))| < \theta \tag{2}$$

where θ is a parameter defining the strictness of the similarity criterion.

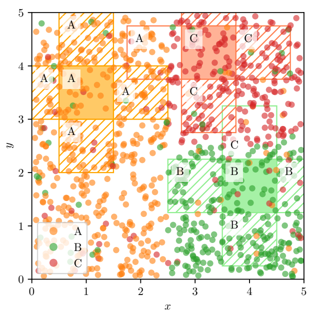
Of course, definition 2 is not suitable to capture the similarity amongst hypercubes characterised by high variability of P. In such a case a more complex solution is required:

$$H_1 \overset{P}{\approx} H_2 \Leftrightarrow \text{mae}(f_{1,2}, H_1 \cup H_2) \leq 0.5(\text{mae}(f_1, H_1) + \text{mae}(f_2, H_2)) \tag{3}$$

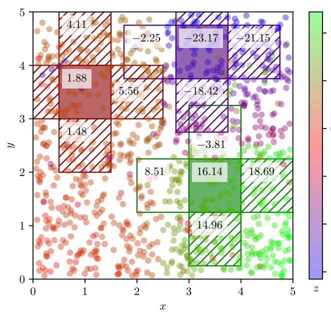
where mae(f, H) is the mean absolute error of the linear function f in approximating P for data in H. In other words, H_1 and H_2 are output-similar if it is possible to *(i)* merge the two hypercubes, *(ii)* find a linear combination $f_{1,2}$ of the input variables representing the input/output relationship of the so merged regions, and *(iii)* reach a predictive performance of $f_{1,2}$ better than the average performance of the linear functions f_1, f_2 associated with the corresponding separated subregions.

For instance, in Figure 1 we report examples of similarity assessments calculated for a generalised extractor applied to a classification task (Figure 1a) and to a regression task (Figures 1b and 1c). Figures concerning the regression task represent constant and non-constant extractor outputs, respectively. The example assumes a 2-dimensional data set with continuous input features both ranging in the interval [0, 5]. In the figures, hypercubes to be expanded/merged are those having coloured backgrounds. Possible adjacent hypercubes to be joined to them are represented as hypercubes having no background. Adjacent hypercubes that are similar to the hypercubes to be expanded are represented with hatched background. It is worth noting that for the example depicted in Figure 1b a similarity threshold θ equal to 5.0 has been chosen. In Figure 1c the predictive errors corresponding to the adjacent hypercubes as well as the calculated errors of the possible merged regions are omitted for the clarity of the image.

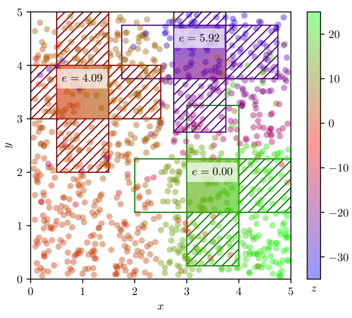
Predictive error assessment A generalised metric is necessary to evaluate the predictive performance of a set of rules for both classifications and regressions. We propose the following



(a) Classification labels.

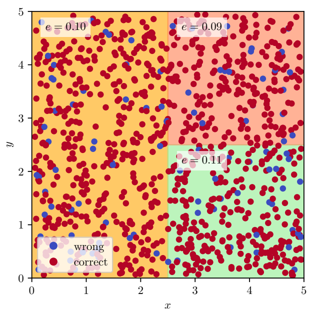


(b) Constant outputs.

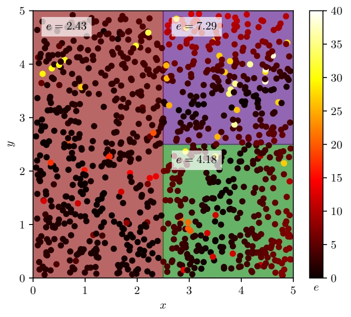


(c) Regression laws.

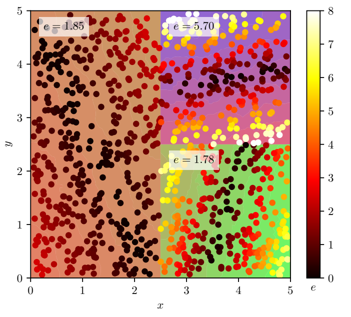
Figure 1: Two-dimensional example of different similarities calculated for generalised extractors. Hatched regions are suitable to be merged with the adjacent one (coloured background).



(a) Classification labels.



(b) Constant outputs.



(c) Regression laws.

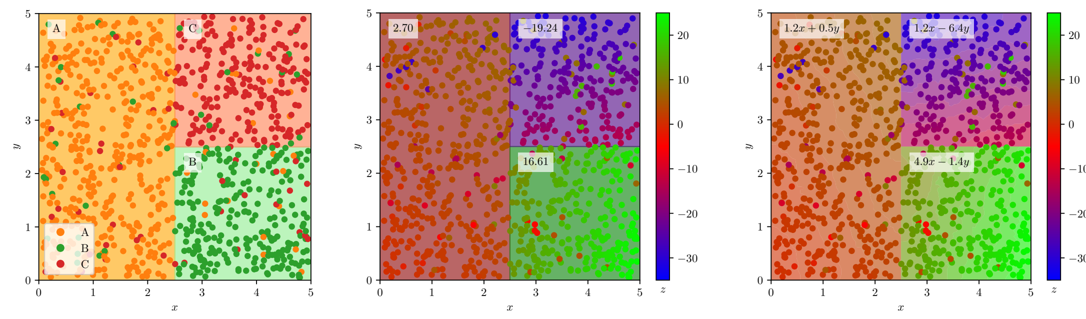
Figure 2: Two-dimensional example of different predictive errors measured for generalised extractors.

function as error function for a rule set R applied to a data set D:

$$\text{error}(R, D) = \begin{cases} \text{mae}(R, D) & \text{(regression)} \\ 1 - \text{accuracy}(R, D) & \text{(classification)} \end{cases} \tag{4}$$

where $\text{mae}(R, D)$ and $\text{accuracy}(R, D)$ are the mean absolute error and the classification accuracy score, respectively, calculated on the output predictions obtained via the rules in R, for the data set D, and w.r.t. the expected outputs for D.

In Figure 2 we report some examples of predictive errors measured for a generalised extractor by assuming a 2-dimensional data set with continuous input features both ranging in the [0, 5] interval. The figure represents a classification task and two regression tasks. The first regression task is approximated by the extractor with constant outputs, whereas the second is associated with non-constant outputs. The predictive error e is reported as misclassifications in the first case and absolute error in the others.



(a) Classification labels. (b) Constant outputs. (c) Regression laws.

Figure 3: Two-dimensional example of different predictions provided by generalised extractors.

3.1.2. Approximating predictions

As for the approximation of output predictions associated with each hypercube, they are usually computed on the basis of the predictions provided by the underlying black box when applied to an *extended* training set. The extended training set may consist of the original data the predictor has been trained upon – or a subset of it –, possibly augmented with some further data. Data augmentation via random input samples is useful to attain higher predictive performance, provided that the predictor is used as an oracle to compute the corresponding expected outputs.

Provided that the input space has been adequately partitioned into several hypercubes, the prediction associated with each hypercube may consist of *(i)* a constant numerical value (e.g., Iter, GridEx) or, *(ii)* a linear combination of the input variables (e.g., GridREx). The latter option, in particular, is well-suited for regression tasks, while the former may support both classification and regression tasks.

To choose the best output value corresponding to each hypercube, one may either *(i)* aggregate the predictions corresponding to all available points in that hypercube – e.g. via the 'mean', 'mode', or 'median' statistical aggregation functions –, or *(ii)* fit a local function *locally* approximating the predictor in that hypercube. Again, which option is better really depends on the learning task the underlying predictor has been designed for.

In Figure 3, we report examples of predictions provided by a generalised extractor. As for the previous examples, a 2-dimensional data set with continuous input features both ranging in the [0, 5] interval is assumed. The Figure shows a classification task and two regression tasks, where the former regression task is approximated by the extractor with constant outputs. The background colour represents the output provided by the extractor.

3.1.3. Output rule set creation

After selecting a set of input space regions and one output decision for each of them, hypercube-based extractors build a set of rules where each one is composed of a precondition and a postcondition. The precondition is a formal description of a single input region in terms of individual features, for instance by means of value inclusion inside an interval. Hypercubic n-

dimensional regions may be described through the conjunction of (at most) n interval inclusion conditions. On the other hand, the postcondition is simply the decision calculated for the region on the basis of the task at hand, as previously described. Thus, extracted human-readable logic rules generally have the following format:

$$\text{Output is } O \text{ if } X_1 \in [l_1, u_1], X_2 \in [l_2, u_2], ..., X_n \in [l_n, u_n],$$

where O is the output decision and $X_1, X_2, ..., X_n$ are input variables assuming values included in the intervals described by corresponding lower-bounds l_i and upper-bounds u_i.

3.2. Comparison of Existing Methods

In the following a comparison between two hypercube-based SKE algorithms – namely Iter and GridEx – is provided to give a practical demonstration of our methodology. Differences in the input space partitioning and in the decision approximation are highlighted in particular. Output rules are lists of logic rules in both cases, following the convention described in Subsection 3.1.3. It worths noting that both algorithms assume input features to be continuous and may be applied to any kind of BB predictor, being pedagogical SKE methods.

Iter The Iter algorithm [13] is based on the iterative creation and expansion of hypercubes inside the input feature space, until a maximum number of iterations is reached or, otherwise, the whole input space is covered. The expansion may terminate also if it is not possible to further expand the hypercubes. In those cases additional cubes may be created to cover the remaining space.

Iter is limited to regression tasks by design, and performs averaging operations to associate output values to hypercubes. For each cube, Iter selects all the training samples inside it and calculates the mean prediction by using the underlying BB as an oracle. If the training samples are not enough to satisfy the minimum amount specified by the user, extra random samples are generated and predicted together with the others.

Iter also takes advantage of a similarity criterion to expand the hypercubes. In particular, at every iteration all the possible expansions around each cube are considered, but only one is performed, i.e., the one capable of expanding a cube towards the most similar input space region. Similarity is calculated via mean absolute difference between the output values of the cubes to be expanded and the eligible cubes around them.

GridEx The GridEx algorithm [14] may be considered as an extension of Iter aimed at overcoming its major drawback, i.e., the non-exhaustivity of its output rules. GridEx achieves this goal because it is exhaustive by design. Unlike Iter, GridEx adopts a top-down approach to split the input feature space into hypercubes. It iteratively partitions the *whole* space according to some defined strategy, marking at each iteration if the created partitions are *negligible* (i.e., they contain no training samples, so they are discarded since it is not relevant to have rules associated to them), *eligible* for further partitioning (if they contain samples that are not enough *similar*), or *permanent* (otherwise, if they contain similar training instances and, thus, these cubes should have a good predictive performance). Strategies to split the input space are *fixed*, if the user specifies for each iteration how many partitions have to be performed along all the input dimensions, or *adaptive*, if the number of splits is determined through the relevance of each input feature w.r.t. the output variable. Since GridEx has been designed exclusively for regression tasks, as ITER, also in this case output decisions are obtained via local averaging calculations and actual regression rules are not supported.

Similarity between samples is assessed through the output value standard deviation of all the instances included inside a hypercube. If the standard deviation is below a user-defined threshold, then the cube only contains similar samples and it is not further partitioned. Otherwise, GridEx attempts to split the cube in smaller regions, possibly enclosing more similar samples. Since the readability of the output model depends on the number of extracted rules, it is of paramount importance to keep it as low as possible. For this reason a merging phase is performed after every splitting iteration as an optimisation to reduce the number of rules. Indeed, adjacent cubes are pairwise merged according to a similarity criterion on the contained samples. The merging phase is iterative: at each step are merged only the two adjacent cubes resulting in the merged hypercube having the lowest standard deviation, and it terminates when it is not possible to further merge cubes without exceeding the standard deviation user-defined threshold.

A first GridEx generalisation supporting regression rules as output decisions has led to the GridREx algorithm [24]. GridREx can extract fully regressive rules, with a linear combination of the input variables as a postcondition. Even though all the other details are identical to GridEx, GridREx is able to achieve better predictive performance, fidelity, and readability than GridEx.

4. Conclusions

In this paper we generalise a class of SKE techniques, namely hypercube-based methods, to make them suitable for classification tasks as well as regression tasks. We do so by proposing a common model for these methods, where algorithmic patterns currently adopted by hypercube-based SKE algorithms are relaxed, in order to widen their applicability scopes. Our future works will focus on extending the proposed generalisation to a wider class of SKE techniques, to enable a higher degree of usability for SKE techniques existing in the literature.

Acknowledgments

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

References

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

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

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

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

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

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

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

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

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

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

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

[12] F. Sabbatini, C. Grimani, Symbolic knowledge extraction from opaque predictors applied to cosmic-ray data gathered with LISA Pathfinder, Aeronautics and Aerospace Open Access Journal 6 (2022) 90–95. URL: https://doi.org/10.15406/aaoaj.2022.06.00145. doi:10.15406/aaoaj.2022.06.00145.

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

[14] F. Sabbatini, G. Ciatto, A. Omicini, GridEx: An algorithm for knowledge extraction from black-box regressors, in: D. Calvaresi, A. Najjar, M. Winikoff, K. Främling (Eds.), Explainable and Transparent AI and Multi-Agent Systems. Third International Workshop, EXTRAAMAS 2021, Virtual Event, May 3–7, 2021, Revised Selected

Papers, volume 12688 of *LNCS*, Springer Nature, Basel, Switzerland, 2021, pp. 18–38. doi:10.1007/978-3-030-82017-6_2.

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

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

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

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

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

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

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

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

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

[24] F. Sabbatini, R. Calegari, Symbolic knowledge extraction from opaque machine learning predictors: GridREx & PEDRO, in: G. Kern-Isberner, G. Lakemeyer, T. Meyer (Eds.), 19th International Conference on Principles of Knowledge Representation and Reasoning (KR 2022), IJCAI Organization, Haifa, Israel, 2022, pp. 554–563. doi:10.24963/kr.2022/57.

[25] R. Setiono, W. K. Leow, J. M. Zurada, Extraction of rules from artificial neural networks for nonlinear regression, IEEE Transactions on Neural Networks 13 (2002) 564–577. doi:10.1109/TNN.2002.1000125.

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

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

[28] R. Konig, U. Johansson, L. Niklasson, G-REX: A versatile framework for evolutionary data

mining, in: 2008 IEEE International Conference on Data Mining Workshops (ICDM 2008 Workshops), 2008, pp. 971–974. doi:10.1109/ICDMW.2008.117.

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

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

[31] P. M. Murphy, M. J. Pazzani, ID2-of-3: Constructive induction of M-of-N concepts for discriminators in decision trees, in: L. A. Birnbaum, G. C. Collins (Eds.), Machine Learning: Proceedings of the Eight International Workshop (ML91), Morgan Kaufmann Publishers, Inc., San Mateo, CA, USA, 1991, pp. 183–187.

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

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

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

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