Tree based Diagnosis Enhanced with Meta Knowledge Applied to Dynamic Systems

Louis Goupil, Louise Travé-Massuyès, Elodie Chanthery, Thibault Kohler, Sébastien Delautier

Tree based Diagnosis Enhanced with Meta Knowledge Applied to Dynamic Systems ⋆

Louis Goupil * **Louise Travé-Massuyès** ** **Elodie Chanthery** **
Thibault Kohler * **Sébastien Delautier** *

* *Atos, Toulouse, France (e-mail: lgoupil@laas.fr, thibault.kohler@atos.net, sebastien.delautier@atos.net)*
** *LAAS-CNRS, ANITI, Université de Toulouse, CNRS, INSA, Toulouse, France (e-mail: louise@laas.fr, echanthe@laas.fr)*

Abstract: This paper presents an online data and knowledge-based diagnosis method. It leverages decision trees where decisions are informed by diagnosis meta knowledge, specifically focusing on the properties of diagnosis indicators. This knowledge is used at each node to articulate a symbolic classification problem, outputting discriminating functions. The outcome is a multivariate decision tree that produces a compact model for diagnosis. The use of decision trees increases the explainability of the outcome, all the more so as one discovers the explicit formal expressions of diagnosis indicators, structured in the form of analytical redundancy relations. This article centers on the essential components required for the method to be suitable for dynamic systems. It is tested on the well-known two-tank system, showing that the performances match those of model-based diagnosis.

Keywords: Fault detection and diagnosis, Symbolic classification, Decision Tree, Analytical redundancy relation, Meta knowledge injection, Symbolic regression, Genetic algorithm.

1. INTRODUCTION

Diagnosis consists in finding *when* a system is behaving abnormally, but also *what* causes a fault. Model-based diagnosis -inferred from system equations- leads to highly efficient solutions but requires extensive knowledge about the system (Slimani et al. (2018)) and suffers from uncertainties inherent to real systems. Data-based diagnosis -inferred from operational data gathered on a system- requires no prior knowledge of systems but rarely grants explanations as to what causes a fault (Tidriri et al. (2016), Lei et al. (2020)). Decision trees (DTs) are data-based methods that can give explainable decisions, with some precautions. This explainability generally requires manipulating conditions returned on the tree paths (Izza et al., 2022a,b). This article proposes a diagnosis algorithm in the form of a DT enhanced with meta knowledge on the properties of diagnosis indicators. It will take advantage of both data-based and model-based techniques, bringing answers both to the *when* and *what* questions.

The most common version of DTs is univariate decision trees (Priyam et al. (2013)). However, system diagnosis is most often obtained by studying relations between measured input and output variables. Thus, univariate DTs may not efficiently address diagnosis problems. That is why we propose a multivariate approach that looks for relations involving several measured variables to discriminate the data in DT nodes as precisely as possible.

⋆ This research is funded by Atos. This project is related to ANITI through the French "Investing for the Future – PIA3" program under the Grant agreement n°ANR-19-PI3A-0004.

This paper main contribution is DT4X, an algorithm that builds a multivariate DT performing explainable diagnosis. The second contribution relies on tree nodes discrimination done by finding relations between observable variables. These relations correspond to diagnosis indicators for they have the same semantics and mathematical form as classical analytical redundancy relations (ARR) from model-based diagnosis. The use of a DT and the identification of diagnosis indicators provide explainability. To sum up, DT4X finds diagnosis indicators for the system and uses them as discriminating relations in a multivariate decision tree. Finally, DT4X is successfully applied on a dynamic system, showing its good performances compared to model-based diagnosis.

Section 2 presents the related works. Section 3 gives the background information required to understand how DT4X works. The algorithm is detailed and its pseudo code given in Section 4. Section 5 demonstrates the application of DT4X to the two-tank use case.

2. RELATED WORK

In De Kleer and Kurien (2003) or Pulido and Alonso (2002), authors show that diagnosis can be accurate and explicable when based on the full model of the system. However, this model is not always accessible and, more often than not, only easy-to-obtain information is available, like which variables are measured and the model structure.

Methods presented in Jung (2022) or Mohammadi et al. (2022) try to exploit this easy-to-obtain system information to perform model-based diagnosis. They conduct structural analysis to derive ARRs and replace model-

based residual generators that require the full model of the system by learned residual generators. They use machine learning algorithms trained on data measured on the system to generate residuals.

On the other hand and despite tremendous advances in explicable AI, methods based only on data are accurate but lack explainability, as discussed in Dwivedi et al. (2023). One exception is DTs that, particularly when of modest size, can be inherently explicable. DTs can be categorized into two types: univariate and multivariate (Yıldız and Alpaydın (2000)). Univariate DTs do not fit well our diagnosis problem (see Section 3.1) given that faulty classes are known to be isolable through multivariate relations (Gao et al. (2015)). Multivariate DTs (Brodley and Utgoff (1995)) allow studying the joint influence of multiple system variables. This is why we advocate them in this paper.

To the best of our knowledge, the only work focusing on applying multivariate decision trees to diagnosis problems in the literature is (Goupil et al., 2023), which limits itself to static systems. This paper goes further by addressing the more difficult case of dynamic systems and presenting a way of taking dynamics into account. It also adds a "refitting" stage to the proposed algorithm, which improves the overall accuracy of the resulting tree.

Hence, this paper aims to apply multivariate DTs to diagnosis. It leverages meta knowledge about diagnosis indicators to extract multivariate discriminating expressions. These expressions happen to have the same properties as model-based ARR.

3. BACKGROUND

3.1 Decision Trees

A Decision Tree $T(E, N)$ is a directed acyclic graph having at most one edge between every pair of nodes. E is the set of edges and N is the set of nodes. T has one root node n_0, characterized by no incoming edge. All other nodes have exactly one incoming edge. Nodes that do not have an outgoing edge are called leaves. We consider only binary decision trees, meaning each node that is not a leaf has exactly two outgoing edges, associated to a node. A path of the tree goes from the root node n_0 to a leaf.

A DT can be used to classify a sample $x = (x_1, \dots, x_n) \in \mathbb{R}^n$. Let us define C the set of possible classes. A dataset $\mathcal{D}$ is a set of pairs (x, l) with $l \in C$. A sample can designate x alone or the whole pair (x, l), depending on the context. When training the DT, a sample is a pair and when using the tree to predict the class of x, x is the sample.

Training, equivalent to building the DT, starts from n_0 and figuratively stores the entire training dataset inside it. Then, if the node is pure, meaning that all samples it contains belong to the same class l, it is considered as a leaf and labeled with that class. If the node is not pure, a discriminating criterion d is computed. The most common algorithms used to determine discriminating criteria are *gini* and *entropy* (Priyam et al. (2013)). Once d is chosen, all samples in the node are evaluated on d leading to the creation of two new nodes: one for the samples that satisfy the criterion ($d(x)$ is true), and one for those that do not ($d(x)$ is false). This process is repeated for newly created nodes until there are no impure nodes left. As a consequence, when the DT is fully trained, each leaf is assigned a class $l \in C$.

Once the DT has been trained, it can be used to predict the class of a sample x. For multivariate DT, a non-leaf node is associated with k features, the value of k being dependent of the node. Each non-leaf node is also associated with a subset of $\mathbb{R}^k$. The decision made at a node depends on whether the actual values of the k features belong to the associated subset.

During the prediction phase, classification of a sample x is performed by following the path of decisions that corresponds to its feature values until reaching a leaf node. The predicted class is the one associated with this leaf node.

Diagnosis can be considered as a classification problem with C containing the nominal behavior class and the faulty classes. A sample x is formed by the observable variables measured on the system at a specific timestep. In the following, $\mathcal{D}$ contains the samples from these classes.

3.2 Diagnosis indicators

A diagnosis indicator is a function that maps quantities that are observable, or those deducible from observations, into a scalar value, providing an indication of a fault.
The value of a diagnosis indicator should be zero, or close to zero, for nominal behavior samples and non-zero for some faulty behavior samples. To *evaluate* a diagnosis indicator $\mathbf{d}$ on a sample x means to compute the image of x by $\mathbf{d}$. A diagnosis indicator evaluated on a nominal sample should always be zero, considering an ideal, non-noisy, environment.

The idea of this paper is to use this knowledge to find appropriate multivariate relations that will be used to take a decision in each node of the diagnosis tree. To do so, we propose to use symbolic classification, modified to incorporate this knowledge and constrain the output function to have the properties of a diagnosis indicator.

3.3 Symbolic Classification

Symbolic classification is a binary classification method. It estimates a function $f : \mathbb{R}^n \to \mathbb{R}$ knowing pairs (x, l) with $x = (x_1, \dots, x_n) \in \mathbb{R}^n$ and $l \in C$, C being a set of classes of cardinality two, $\{0, 1\}$. f is then used together with a *classification function* $t : \mathbb{R} \to [0, 1]$ to predict the class of a sample x. The output of t is in $[0, 1]$ but a threshold can be applied to obtain the class in $\{0, 1\}$. t is known. Thus, applying a threshold to $t(f(x))$ gives the class of x. This estimation of f is done without assuming the structure of f and discovers a precise analytical solution. It relies on a genetic algorithm that takes as inputs a set of pairs $\mathcal{D} = \{(x, l)\}$ called the dataset and a set of operators O (e.g. $+, *, -, /, \sqrt{}, ||, \log$, etc.). It searches for the best combination $c_{x,O}$ of variables $(x_1, \dots, x_n)$ and operators so that $c_{x,O} = f(x)$. The genetic algorithm generates candidate functions $c : \mathbb{R}^n \to \mathbb{R}$. These candidate functions are then evaluated using a fitness function. Since a symbolic classifier aims at

predicting classes, the fitness function used is most often the log loss function (Bishop (2006)), given by:

$$Fitness(c) = -\frac{1}{|\mathcal{D}|} \sum_{(x,l)\in\mathcal{D}} [l \ln(t(c(x))) + (1-l)\ln(1-t(c(x)))]$$

Training symbolic classification corresponds to generating candidate solutions c until reaching one that is *close enough* to f, *close enough* being defined by a given threshold. The python package `gplearn` (Stevens (2016)) provides an implementation. The classification function t can take many shapes. Its default value in `gplearn` is a sigmoid function. In the context of this paper, the classification function is customized to fit our needs, see Section 4.5.

4. DT4X

DT4X stands for Diagnosis Tree for eXplainability with 4 main features: multivariate analysis, explicable decision-making, incorporation of meta knowledge and use of symbolic classification.

4.1 Principle

DT4X aims at building (training) a binary DT to diagnose a system. Its pseudo-code is given in Algorithm 1. The inputs for DT4X are the training dataset $\mathcal{D}$, the operators and the hyper-parameters (see Section 4.3) values.
$\mathcal{D}$ contains pairs of corresponding (x, l) with $x \in \mathbb{R}^n$, the observable variables and $l \in C$ the diagnosis, with C the set of diagnosis classes $(nominal, faulty_1, \ldots, faulty_m)$. A faulty scenario can be any faulty state of the system, including multiple faults occurring at the same time.
The operators are a set of functions specified by the user. The usual operators are: $+, *, -, /, sign, abs, \sqrt{}, cos, sin, ...$

In the resulting DT, each node $n_i \in N$ that is not a leaf contains a diagnosis indicator $\mathbf{d}_{n_i} : \mathbb{R}^n \to \mathbb{R}$. Each diagnosis indicator $\mathbf{d}_{n_i}$ is used to partition the data into two disjoint subsets, depending on whether $t(\mathbf{d}_{n_i}(x)) = 0$ or $t(\mathbf{d}_{n_i}(x)) = 1$ for x contained in n_i. The two subsets are then sent to a different child node. Each leaf of the resulting tree has a label that is the classes predicted for the data reaching this leaf.

The algorithm uses concepts that need to be defined, with X_p and X_r hyper-parameters of DT4X:

Definition 1. (Pure with *label*). A node n_i is said to be pure with *label* if at least X_p% of the samples belonging to n_i are of class *label*.

Definition 2. (Relevancy in n_i). A class is said to be relevant in n_i if the amount of samples of this class constitutes at least X_r% of the whole dataset.

4.2 Algorithm

Algorithm 1 gives the pseudo-code of DT4X. The arrow symbol with a plus ($\leftarrow$ +) means that the value is appended to the variable.

During the training phase, a node $n_i \in N$ contains a subset of samples $\mathcal{D}_{n_i} \subset \mathcal{D}$. Each sample (x, l) contained in n_i verifies the conditions defined on the edges leading to n_i from the root node. At the beginning of DT4X (line 1), the root node *rootNode* contains the entire set $\mathcal{D}$. DT4X builds the tree starting from the root node and then going through every single node in their order of creation. The algorithm stops when no nodes are left (line 2).

Algorithm 1 DT4X

Input: Training Dataset, Operators, Hyper-parameters
Output: DT with Diagnosis Indicators

```
1:  currentNodes ← rootNode
2:  while currentNodes is not empty do
3:      for all node ∈ currentNodes do
4:          if node is pure with label then
5:              node is leaf
6:              node ← label
7:          else
8:              pairsToTry ← generate pairs
9:              pair ← first element of pairsToTry
10:             while not check foundExpression
11:                   and pairsToTry not empty do
12:                 balance pair
13:                 foundExpression ← SC on pair
14:                 pair ← next element of pairsToTry
15:             end while
16:             if foundExpression then
17:                 lNode, rNode ← split according to
18:                                foundExpression
19:                 futureNodes ← +lNode, rNode
20:             else
21:                 node is leaf
22:                 node ← majority label
23:             end if
24:         end if
25:     end for
26:     currentNodes ← futureNodes
27: end while
```

When reaching a node n_i, DT4X first checks whether n_i is `pure with` *label* (line 4) (see Definition 1). If it is the case, n_i is designated as a leaf and the label *label* is associated with it (lines 5 and 6).
Otherwise, the goal is to find a new diagnosis indicator $\mathbf{d}_{n_i}$ that splits the data belonging to $\mathcal{D}_{n_i}$ (line 8 to 15). Thus, let us `generate` a set of possible pairs of classes (*pairsToTry*) to split using a symbolic classifier (line 8).

Two cases are distinguished for the pair generation.

If the nominal class is relevant (see Definition 2) in the node n_i, the set of pairs to try (*pairsToTry*) is built by having the nominal data as the first class, and any of the faulty classes relevant in n_i as the second class. This means pairs of the shape $(nominal, faulty_k)$.

If the nominal class is not relevant in n_i, the first step is finding the set of faulty classes that are relevant in n_i. Then, all permutations of pairs of these classes are generated. For p faulty classes, this results in $p * (p-1)$ pairs. It also means that if the pair $(faulty_i, faulty_k)$ is present, the pair $(faulty_k, faulty_i)$ will be present too. This is important because of the following step: the first class of the pair is then modified. During the balancing process (see further), half of the data of the first class is made of samples of the class itself, and the other half is

made of nominal data randomly selected from the initial dataset. This ensures that symbolic classification finds an expression that is worth 0 for both nominal samples and samples belonging to the first class of the pair. This expression will also be different from 0 for samples of the second class of the pair. In other words, the expression is triggered by samples of the second class but not by samples of the first or by nominal samples.

Once the set of pairs (*pairsToTry*) is generated, the algorithm loops over those pairs until either it runs out of pairs to try or until a diagnosis indicator $\mathbf{d}_{n_i}$ is found that discriminates the classes from the pair correctly (lines 10 to 11).
When a pair is selected from *pairsToTry*, step one is to `balance` samples in the two classes (line 12). This preprocessing checks which class in the pair has fewer samples and randomly selects the same number of samples from the class that has the most samples. This is done to ensure symbolic classification is performed with balanced classes.

Then, the symbolic classification algorithm is performed on the pair as described in Section 3.3 (line 13). Symbolic Classification (`SC` in the pseudocode) always returns a candidate expression, named $foundExpression$, that is the best expression found w.r.t. the fitness. However, while this expression might be the best found, it might not necessarily qualify as a good diagnosis indicator, either because the best possible expression has not been found or because the pair of classes used to train the symbolic classification are samples from non isolable fault cases. Thus, it is important to `check` (line 10) if the found expression is a valid diagnosis indicator. This is done by taking two consecutive tests **T1** and **T2**, with hyper-parameters X_{T_1} and X_{T_2}.
T1 checks that the nominal data from the whole dataset is predicted as 0 by $foundExpression$. If at least $X_{T_1}\%$ of the nominal data is predicted as 0 then the test is passed successfully.
T2 checks that $foundExpression$ predicts correctly $X_{T_2}\%$ of the data used to find it through symbolic classification. This percentage does not include data discarded during the balancing process. This test ensures that $foundExpression$ is classifying correctly.
If either **T1** or **T2** is false, then $foundExpression$ is not considered as a valid diagnosis indicator and the loop over the pairs continues.

Once either a valid diagnosis indicator $\mathbf{d}_{n_i}$ has been found or all pairs have been tested, the while loop is exited. If a diagnosis indicator $\mathbf{d}_{n_i}$ was found, corresponding to $foundExpression$ (line 16), the data in node n_i is `split` according to $\mathbf{d}_{n_i}$. The algorithm evaluates $\mathbf{d}_{n_i}$ on the samples within $\mathcal{D}_{n_i}$. If the result is 0, the sample (x, l) is sent to the left child of the current node ($lNode$). If the result is different from 0, the sample is sent to the right child ($rNode$). If no diagnosis indicator was found (line 20), the class with the most samples in $\mathcal{D}_{n_i}$ has the `majority`, and the node is labeled with this class (line 22) and declared a leaf.

4.3 Parameterizing DT4X

The hyper-parameters are summarized in Table 1. The DT4X hyper-parameters are described in Section 4.2.

DT4X	Default Value
purity threshold X_p	0.95
relevance threshold X_r	0.001
performance on nominal threshold X_{T_1}	0.95
indicator performance threshold X_{T_2}	0.90
Symbolic Classification	**Default Value**
ϵ	0.01
population size	5000
maximum number of generations	50
proportion of samples used	1
parsimony coefficient	0.02

Table 1. List of hyper-parameters and their default value

The ϵ parameter (Section 4.5) has a powerful influence on the outcome of symbolic classification, so it should be modified according to the studied system. It should be scaled according to the order of magnitude of the data. The *population size* parameter is the number of candidate solutions generated at each generation. A higher value reduces the number of generations before reaching a good solution but it extends training time. The *maximum number of generations* is the number of generations beyond which the algorithm stops, even if it has not found a solution. High value means more chances to find the right solution, but when no solution is to be found, it will lengthen the time it takes to stop. The *proportion of samples used* is the dataset fraction used to test each candidate solution. It provides a trade-off between computation time and accuracy. Higher accuracy means finding better diagnosis indicators, leading to faster predictions. Indeed, prediction time should have priority over training time. Thus, the whole dataset is used by default. When computing fitness for a candidate solution, a penalty is subtracted from its score. This penalty is the *parsimony coefficient* multiplied by the expression length of the candidate solution, favoring shorter solutions.

4.4 Prediction with DT4X

Once DT4X has built the tree, it can be used to predict the class of a sample x, i.e. to diagnose the system status when x is (or was) measured. Prediction is performed by inputting x at the root node. When x reaches a node n_i, the diagnosis indicator $\mathbf{d}_{n_i}$ of that node is evaluated on x. Similarly to the process described in Section 3.1, depending on the result of this evaluation, x is sent to one output edge or the other. When x reaches a leaf node, the prediction is determined by the label of that leaf node.

4.5 DT4X Classification Function

The classification function t for symbolic classification is customized for discovering diagnosis indicators:

$$\text{If } |y| < \epsilon, t(y) = 0, \text{ otherwise } t(y) = 1, \qquad (1)$$

with ϵ a DT4X parameter. If nominal data is inputted as class 0 and a faulty scenario data as class 1, and a function is discovered with high accuracy on this data, then this function is a diagnosis indicator, as it is characterized by being null for nominal cases, non-null for the class 1 faulty scenario, and involving only observable variables. Such indicator is sensitive to, at least, the fault used to find it.

5. DT4X EXTENDED FOR DYNAMIC SYSTEMS AND IMPROVED ACCURACY

5.1 Refitting

Once the DT is fully grown, it can sometimes be further improved. Indeed, DT4X relies on a genetic algorithm that, while very likely, does not guarantee convergence. Reusing symbolic classification on a pair that initially did not produce a diagnosis indicator might produce one with a different seed for the randomization of mutations. Thus, a `refit` function has been designed to automatically select leaf nodes with the lowest purity scores and retry symbolic classification with all the relevant classes in those nodes. This iterative process has shown to improve the tree accuracy (see Section 5.4). The `refit` function also allows adjustments to symbolic classification parameters, such as the mutation rate or the population size. `refit` is usually performed only once because after two tries, an impure node will most likely contain classes that are non-isolable from each other. It has not been tested yet, but future work could use `refit` to retrain part of the tree following, for instance, a concept drift in the dataset.

5.2 Dynamic Systems

DT4X trains from samples corresponding to a specific timestep. In order to work with dynamic systems, derivatives (and/or integrals) of the variables must be given as input along the variables themselves (Mohammadi et al. (2023)). They contain information about the system evolution. This implies prior knowledge of the highest derivative order, needed to build relevant diagnosis indicators. In the current implementation, computation of the derivatives is performed for all the continuous domain observable signals of x in the data preprocessing stage and added to the feature vector given as input to DT4X.

One could think that a better solution is to implement a derivative operator to give to the symbolic classification phases of DT4X. It would allow candidate solutions to contain any order derivatives and to compose the derivative operator and other operators. However, this requires information about the neighbor samples that must then be stored and this requires computing the derivatives at training phase, which would be very expensive. Also, deciding the size of the window of samples to give to DT4X is as complex as deciding the maximum derivative order.

5.3 Data-based Analytical Redundancy Relations

Diagnosis indicators found by DT4X have the same properties as model-based ARR but they are computed from data. Hence, we define the specific diagnosis indicators found along the DT and name them *data-based ARR* as they are obtained from a dataset $\mathcal{D}$ whereas ARRs are obtained from a system model. Consequently, data-based ARR have a validity domain limited to the data set.

Definition 3. (Data-based ARR for a dataset $\mathcal{D}$). Consider a dataset of pairs $\mathcal{D} = \{(x, l)\}, l \in C$. $C_0 \in C$ is the class of the nominal samples. The relation of the form $\mathbf{d}(x', \dot{x}', \ddot{x}', ...) = r$, with input x' a subvector of x and output r, a scalar named *residual*, is a *data-based ARR* for the dataset $\mathcal{D}$, denoted $ARR_{\mathcal{D}}$, if, for all x such that there exists a pair $(x, C_0) \in \mathcal{D}$, it holds that $r = 0$.

5.4 The Two-Tank System case study

DT4X has been tested on a two-tank dynamic system presented in B.Ould Bouamama (2001), shown in Fig. 1.

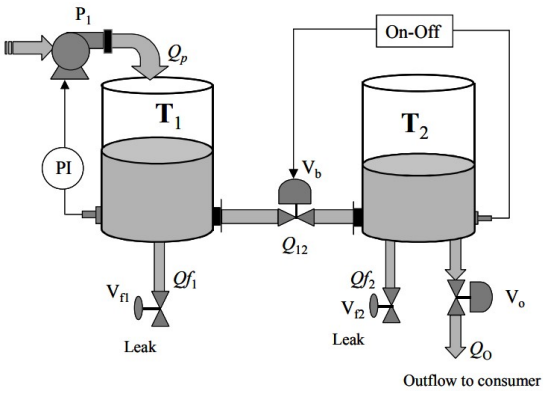

Fig. 1. Scheme of the Two-Tank system

This system is susceptible to 12 possible faults. This means 13 possible situations including the nominal case. For the sake of this study, faults can not occur simultaneously.

8 observable variables are in the system, including a binary one. Thus, only 7 derivatives can be computed, using the `gradient` method of the NumPy package. Hence, the dataset contains samples of 13 feature values associated to a label $l \in C$ with C the set of possible diagnoses.

Data-Based Results The experiment uses data from a Matlab Simulink [1] model of the system with a hundred 250-second simulations per fault type. The start of the fault is randomly selected between 1 and 200 s. For leaks, the flow rate is also randomized, following a normal distribution around a standard value.

The dataset is split between a training set with 258960 samples, corresponding to 1040 simulations, and a test set with 64740 samples, corresponding to 260 simulations. The training set is then fed to DT4X with default hyperparameters, except for $\epsilon = 1e^{-4}$ and the parsimony coefficient set to 0.003. The operators given to the symbolic classification are $+, -, *, /, \sqrt{}$ and $||$. They are chosen according to expert knowledge about the system. The obtained DT is fine-tuned by refitting the most impure leaf nodes.

DT4X Results The successive DTs obtained by DT4X are available on GitHub [2], along with the detailed list of input variables and faulty scenarios. The accuracy of the resultant DT on the test set is 95.83%. The confusion matrix is shown in Fig. 2. It took 27.23 sec to predict the labels of all samples in the test set, meaning 0.42 ms per sample on average. It was measured on an AMD Ryzen 9 6900hx, 16 cores, using no GPU.

[1] https://cs2ac.upc.edu/en/training-benchmarks/cyber-attacks-benchmark-simulator-1

[2] https://github.com/LGpro/DT4X-on-TwoTank-System

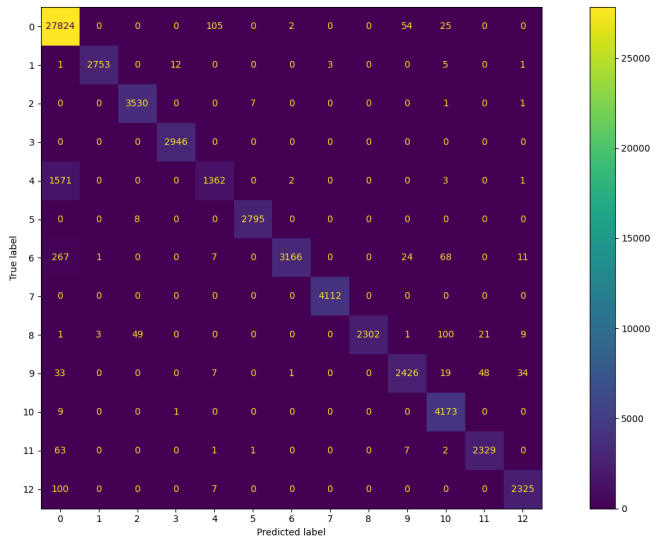

Fig. 2. Confusion Matrix of the Obtained DT

Classical DT Results In order to compare DT4X to the more common univariate DT, a scikit-learn [3] default decision tree (SklDT) was trained on the same dataset, but this time on a GPU. The accuracy of this SklDT on the test set is 96.07%. Prediction time for the 64740 test samples for the SklDT is 0.01s, 2000 times faster than DT4X. However, this SklDT does not find diagnosis indicators and does not give the diagnosability of the system.

Model-Based Results Using the model of the system, it is also possible to compute a set of ARRs. These ARRs have an accuracy of 96.20% on the same dataset. They mostly fail to diagnose the leaks with varying flow rate because they are designed around a fixed threshold that fails to trigger on all possible leak rate values. Similarly, the DT4X tree fails to predict the input flow sensor faults, probably for the same reasons. Similar patterns can be observed between these two methods, due to the fact that they both build diagnosis indicators, which are triggered according to thresholds. However, DT4X only requires data from the system, whereas computing the model-based ARR requires the full physical system model.

6. CONCLUSION

The proposed algorithm generates a highly interpretable multivariate decision tree for diagnostic purposes, utilizing multivariable relations discovered through symbolic classification, which take the form of ARRs (Analytical Redundancy Relations). Tested on the well-known two-tank system, it demonstrates high accuracy. The data-based ARR sometimes exactly coincides with the model-based ARR. This is the case for the well-known polybox example. However, when this is not the case, their interpretability requires further investigation. Future work will focus on evaluating and understanding the discovered diagnosis indicators.

REFERENCES

Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer. P. 209.

B.Ould Bouamama, R. Mrani Alaoui, P.T..M.S. (2001). Diagnosis of a two-tank system. *Intern Report of CHEM-project, USTL, Lille, France*.

Brodley, C.E. and Utgoff, P.E. (1995). Multivariate decision trees. *Machine Learning*, 19(1), 45–77.

De Kleer, J. and Kurien, J. (2003). Fundamentals of model-based diagnosis. *IFAC Proceedings Volumes*, 36(5), 25–36.

Dwivedi, R., Dave, D., Naik, H., Singhal, S., Omer, R., Patel, P., Qian, B., Wen, Z., Shah, T., Morgan, G., et al. (2023). Explainable AI (XAI): Core ideas, techniques, and solutions. *ACM Computing Surveys*, 55(9), 1–33.

Gao, Z., Cecati, C., and Ding, S.X. (2015). A survey of fault diagnosis and fault-tolerant techniques, part I: Fault diagnosis with model-based and signal-based approaches. *IEEE Trans. on Industrial Electronics*, 62(6).

Goupil, L., Chanthery, E., Travé-Massuyès, L., and Delautier, S. (2023). Tree based diagnosis enhanced with meta knowledge. In *34th International Workshop on Principles of Diagnosis (DX'23)*.

Izza, Y., Ignatiev, A., and Marques-Silva, J. (2022a). On tackling explanation redundancy in decision trees. *Journal of Artificial Intelligence Research*, 75, 261–321.

Izza, Y., Ignatiev, A., Narodytska, N., Cooper, M.C., and Marques-Silva, J. (2022b). Provably precise, succinct and efficient explanations for decision trees. *arXiv preprint arXiv:2205.09569*.

Jung, D.E. (2022). Automated design of grey-box recurrent neural networks for fault diagnosis using structural models and causal information. In *Conference on Learning for Dynamics & Control*.

Lei, Y., Yang, B., Jiang, X., Jia, F., Li, N., and Nandi, A.K. (2020). Applications of machine learning to machine fault diagnosis: A review and roadmap. *Mechanical Systems and Signal Processing*, 138, 106587.

Mohammadi, A., Krysander, M., and Jung, D.E. (2022). Analysis of grey-box neural network-based residuals for consistency-based fault diagnosis. *IFAC-PapersOnLine*.

Mohammadi, A., Westny, T., Jung, D., and Krysander, M. (2023). Analysis of numerical integration in RNN-based residuals for fault diagnosis of dynamic systems. *arXiv preprint arXiv:2305.04670*.

Priyam, A., Abhijeeta, G.R., Rathee, A., and Srivastava, S. (2013). Comparative analysis of decision tree classification algorithms. *International Journal of current engineering and technology*, 3(2), 334–337.

Pulido, B. and Alonso, C. (2002). Possible conflicts, arrs, and conflicts. In *DX'02 12th International Workshop on Principles of Diagnosis*.

Slimani, A., Ribot, P., Chanthery, E., and Rachedi, N. (2018). Fusion of model-based and data-based fault diagnosis approaches. *IFAC-PapersOnLine*, 51, 1205–1211. doi:10.1016/j.ifacol.2018.09.698.

Stevens, T. (2016). GPlearn. *Github*. (https://gplearn.readthedocs.io/en/stable/intro.html).

Tidriri, K., Chatti, N., Verron, S., and Tiplica, T. (2016). Bridging data-driven and model-based approaches for process fault diagnosis and health monitoring: A review of researches and future challenges. *Annual Reviews in Control*, 42, 63–81.

Yıldız, O.T. and Alpaydın, E. (2000). Univariate and multivariate decision trees.

[3] https://scikit-learn.org/stable/modules/tree.html