Tree based diagnosis enhanced with meta knowledge

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

Tree based diagnosis enhanced with meta knowledge
Diagnosis indicators discovery as a byproduct

Louis Goupil[1,2], **Elodie Chanthery**[1], **Louise Travé-Massuyès**[1] and **Sébastien Delautier**[2]
[1]LAAS-CNRS, ANITI, Université de Toulouse, CNRS, INSA, Toulouse, France
e-mails: lgoupil@laas.fr, echanthe@laas.fr, louise@laas.fr
[2]Atos, Toulouse, France
e-mail: sebastien.delautier@atos.net

Abstract

This paper presents an online data and knowledge based diagnosis method. It leverages decision trees in which decisions are made based on diagnosis meta knowledge, namely knowledge about the properties of diagnosis indicators. This knowledge is used at the level of each node to set a symbolic classification problem that brings out discriminating functions. This results in a multivariate decision tree that produces a compact model for diagnosis. The use of decision trees increases the explicability of the results found, all the more so as one discovers the explicit formal expressions of diagnosis indicators in the process. The method has been tested on static systems. On the well-known polybox, the three diagnosis indicators known as analytical redundancy relations, that are generally computed from the model, are found.

1 Introduction

Diagnosis aims at finding *when* a system is behaving abnormally, but also *what* causes a fault, i.e. what component is faulty.

On one hand, model-based diagnosis, inferred from system equations, leads to highly efficient solutions, but requires extensive knowledge about the system [1] and often suffers from the uncertainty inherent to real systems. On the other hand, data-based diagnosis, inferred from data gathered on a system in operation and requiring no prior knowledge of systems, rarely grants explanations as to what causes a fault to occur [1]. Most data-based solutions do not provide enough information on what causes the fault, they are often black-box [2]. However, decision trees are data-based methods that still give an explanation for their decision. In this article, we propose a diagnosis algorithm that takes the form of a decision tree enhanced with meta knowledge about the properties of diagnosis indicators. This solution will take advantage both of the data-based and model-based techniques, being able to answer both the *when* and *what* questions.

The most common version of decision trees is univariate decision trees [3]. However, diagnosis of a system is most often obtained by studying relations between exogenous variables (the inputs of the system) and endogenous observable variables (the outputs of the system). Thus, univariate decision trees do not seem to fit diagnosis problems. That is why we propose a multivariate approach that looks for relations involving several observable variables to discriminate the data in decision tree nodes as precisely as possible.

This paper main contribution is DT4X, an algorithm that builds a multivariate decision tree that performs explicable diagnosis. The second contribution is that discrimination in the nodes of the tree is done by automatically finding relations between observable variables that correspond to diagnosis indicators used in model-based diagnosis. To sum up, the new proposed algorithm finds diagnosis indicators of the system diagnosed and uses them as discriminating relations in a multivariate decision tree.

This paper first presents the related works and the background required to understand how DT4X works. Then, the algorithm is detailed and its pseudo code is given. After that, DT4X is applied to the polybox use case and the results are discussed.

2 Related Work

Works such as [4] or [5] have shown 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. For instance, which variables are measurable. Sometimes, the whole structural model of the system is accessible.

Methods such as [6] or [7] try to exploit this easy-to-obtain system information to perform model-based diagnosis. They perform structural analysis (an analytical redundancy approach) and replace the step where full model of the system is required. Instead, they use data-based algorithms trained on data measured on the system. These algorithms output residuals. They actually replace the model-based residual generators.

On the other hand and despite the tremendous advances in explicable AI [8], methods based on data only are accurate but rarely explicable [9], one exception being decision trees when their size remains small enough. Only then can decision trees be considered intrinsically explicable. We can split decision trees into two categories, univariate and multivariate [10]. Univariate decision trees do not fit our problem (see Sections 3.1 and 5.2) because it is known that, in a diagnosis problem, the classes are characterized by multivariate relations [11]. Multivariate decision trees [12] allow to study the joint influence of multiple system variables. To the best of our knowledge, there is no paper focusing on applying multivariate decision trees to diagnosis.

Hence, this paper looks to apply multivariate decision trees to diagnosis, but also to add easy-to-obtain model-based information into the tree, while also using meta-knowledge about diagnosis indicator to improve the multivariate discriminating expression found. As a consequence, the found multivariate expressions happen to be diagnosis indicators themselves.

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 a 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. A path of the tree goes from the root node to a leaf. The children of a node are the nodes reachable from it by following the two outgoing directed edges.

A decision tree can be used to classify a sample x with n features $x = (x_1, x_2, ..., x_n) \in \mathbb{R}^n$. Let us define C the set of possible classes of cardinality m. A dataset is a set of pairs (x, C_i) with $C_i \in C$, $i \in [\![1, m]\!]$. A sample can designate x alone or the whole pair (x, C_i), depending on the context. When training (building) the decision tree, a sample is a pair and when using the tree to predict, the sample is only x.

Training (that is equivalent to building the decision tree) is performed by starting from the lone root node of the tree by figuratively storing all the dataset (all the training samples) inside it. Then, if the node is pure, meaning that samples contained in it are all of the same class C_i, the node is considered a leaf and labeled with the class C_i. 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* [3]. Once d is chosen, all samples contained in the node are tested on d and two new nodes are created, one for the samples that respect the criterion ($d(x)$ is true), and one for those that do not ($d(x)$ is false). Then, the whole process is repeated for nodes that have been created until there are no impure nodes left. As a consequence, when the decision tree is fully trained, each leaf has been given a class C_i of C, $i \in [\![1, m]\!]$.

Once the decision tree has been trained, it can be used to predict the class of a sample x.

For univariate decision trees, each non-leaf node is associated with a feature x_i with $i \in [\![1,\ n]\!]$ and a subset S_i of $\mathbb{R}$. One of the outgoing edges is associated with the condition ($x_i \in S_i$), the other one to values that are outside S_i ($x_i \notin S_i$). This is the criterion mentioned before. A decision in a node is taken based on whether the actual feature x_i belongs to the subset S_i. If it does, it is sent to the edge associated with this condition, if not, to the other.

For multivariate decision trees, a non-leaf node is associated with k features, the value of k being dependent of the node. It is also associated with a subset of $\mathbb{R}^k$. A decision in a node is taken based on whether the actual values of the k features belong to the 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 the set of diagnosis classes and the features of x the observable variables measured on the system.

3.2 Diagnosis Indicators

A diagnosis indicator is an analytical expression of observable variables. More precisely, a diagnosis indicator $\mathbf{d} : \mathbb{R}^n \to \mathbb{R}$ is a function of the n observable variables $x = (x_1, x_2, ..., x_n) \in \mathbb{R}^n$ whose value is zero for nominal samples and non zero in some faulty situations. 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 based on symbolic regression, and it is necessary to understand symbolic regression to be able to understand symbolic classification. The principle of symbolic regression is detailed in [13].

Symbolic Regression

Symbolic regression is a method to estimate a function $f : \mathbb{R}^n \to \mathbb{R}$ knowing pairs $(x, f(x))$ with $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ and $f(x) \in \mathbb{R}$. It is a kind of regression that does not assume the structure of the solution and discovers a precise analytical solution. It relies on a genetic algorithm to find this solution. It takes as inputs a set of pairs $\mathcal{D} = \{(x, f(x))\}$ 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 $(x_1, x_2, ..., x_n)$ and operators so that $C_{x,O} = f(x)$. The genetic method is more precisely described and explained in [13]. The python package `gplearn` [14] provides an implementation.

Symbolic regression algorithms such as the one implemented in `gplearn` accept many hyper-parameters. For instance, the mutation frequency and likeliness for each possible type of mutation, the number of candidates solution at each generation, etc. A major parameter is the fitness function. At each generation, each candidate solution $c : \mathbb{R}^n \to \mathbb{R}$ (a combination of variables $(x_1, x_2, ..., x_n)$ and operators) is tested on the dataset $\mathcal{D}$ and given a score that represents how well it fits f. This score is called the fitness. It is determined using the formula:

$$Fitness(c) = \frac{1}{n} \sum_{(x, f(x)) \in \mathcal{D}} (f(x) - c(x))^2$$

with n being the cardinality of $\mathcal{D}$.

Symbolic Classification

Symbolic classification is a binary classification method to estimate a function $f : \mathbb{R}^n \to \mathbb{R}$ knowing pairs (x, l) with $x = (x_1, x_2, ..., x_n) \in \mathbb{R}^n$ and $l \in C$, C being a set of classes of cardinality two. For the sake of practicality, we choose $C = \{0, 1\}$. Just like symbolic regression, symbolic classification does not assume knowing the structure of the solution, discovers a precise analytical solution and relies on a genetic algorithm to train. It also takes a set of

pairs $\mathcal{D} = \{(x, l)\}$ called the dataset and a set of operators O (e.g. $+, *, -, /, \sqrt{}, ||, \log$, etc.) as input. It also searches for the best combination $C_{x,O}$ of variables $(x_1, x_2, ..., x_n)$ and operators so that $C_{x,O} = f(x)$. Everything is identical to symbolic regression except the set of values for the labels l and the way to compute the fitness. Indeed, the candidate functions, when applied to each sample x, give a number in $\mathbb{R}$. Symbolic classification accepts an additional parameter (compared to symbolic regression) often called the *transformer* $t : \mathbb{R} \to [0, 1]$, a function that gets fed a number $y = c(x) \in \mathbb{R}$ and that outputs $t(y) \in [0, 1]$. This transformer can take many shapes. Its default value in `gplearn` [14] is a sigmoid function. Since a symbolic classifier aims at predicting classes, the fitness function used is most often the log loss function ([15]). It is determined using the formula:

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

with n being the cardinality of $\mathcal{D}$.

In the context of this paper, the transformer is customized to fit our needs, see Section 4.6

4 DT4X

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

4.1 General Principle

The DT4X algorithm aims at building (and training simultaneously) a binary decision tree to diagnose a system. This decision tree is built starting from the root node and by propagating the training data in the corresponding nodes, similarly to what is done traditionally (see Section 3.1). The pseudo code for the DT4X algorithm is shown in Algorithm 1. The inputs for DT4X are the learning dataset $\mathcal{D}$, the operators and the hyper-parameters values.

The dataset $\mathcal{D}$ contains pairs of corresponding (x, l) with $x \in \mathbb{R}^n, n \in \mathbb{N}$, the observable variables and $l \in C$ the diagnosis with C the set of possible diagnosis classes $(nominal, faulty_1, faulty_2, ..., faulty_m)$. A faulty scenario can be any faulty state of the system, even multiple faults occurring at the same time. The dataset must contain data from the nominal class and from all the faulty classes. Indeed the tree will only be able to predict faults used to build it.

The operators are a set of functions specified by the user. The usual operators are: $+, *, -, /, sign, abs, \sqrt{}, cos, sin$, ... By default, the operators are $+, -, *, /$. They should be chosen according to knowledge about the system behavior. For instance, if it is known that a component squares the input to give the output, it makes sense to include the square operator and its inverse, the square root operator.

The hyper-parameters are described in Section 4.4.

In the resulting decision tree, each node $n_i \in N$ that is not a leaf contains a binary diagnosis indicator $\mathbf{d}_{n_i} : \mathbb{R}^n \to \mathbb{R}$. For a subset S^C of C that contains the nominal class, $\mathbf{d}_{n_i}(x_i) = 0$ and for $x_i \in C \backslash S^C$, $\mathbf{d}_{n_i}(x_i) \neq 0$.

Each diagnosis indicator $\mathbf{d}_{n_i}$ is used to split data into two disjoint subsets. Each subset is then sent to a different child node. Each leaf of the resulting tree has a label that is the class predicted for the data that reaches this leaf.

The algorithm uses some concepts that need to be defined.

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.*

The value X_p is a hyper-parameter of the algorithm.

Definition 2 (Representative)**.** *Let us say the majority class (the one with the most samples in n_i) has N samples. A class is said to be in a representative amount in n_i if at least X_r% of the N samples of this class belong to the node n_i.*

The value X_r is a hyper-parameter of the algorithm.

4.2 DT4X Pseudo Code

Algorithm 1 gives the pseudo-code of DT4X. The arrow symbol with a plus ($\leftarrow +$) means that the value is appended to the variable. All the keywords used are explained in Section 4.3.

Algorithm 1 DT4X

Input: Training Dataset, Operators, Hyper-parameters
Output: Decision Tree 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
```

4.3 Pseudo Code In-depth Explanation

During the learning phase, a node $n_i \in N$ contains a set of samples $\mathcal{D}_{n_i} \subset \mathcal{D}$. Each sample (x, l) in n_i verifies the conditions that are defined on the edges that lead to this node from the root node.

At the beginning of the DT4X algorithm (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 the order they are created. The algorithm stops when there are no nodes left to deal with.

When reaching a node n_i, the algorithm DT4X first checks whether n_i is `pure with` $label$ (line 4). If it is the case, the node n_i is designated as a leaf and the label $label$ is associated to it (line 6).

Otherwise, the goal is to find a new diagnosis indicator $\mathbf{d}_{n_i}$ that splits the data belonging to node n_i (line 8 to 15). Since symbolic classification allows classification between two classes only, the algorithm looks for a diagnosis indicator that splits a specific pair of classes. Thus, we need to `generate` a set of possible pairs of classes ($pairsToTry$) to split (line 8).

Two cases are distinguished for the pair generation:

- If there are still nominal samples in a representative amount (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 present in a representative amount as the second class. This means pairs of the shape $(nominal, faulty_k)$.
- If the nominal samples are not present in a representative amount in the node, the first step is finding the set of faulty classes that are present in a representative amount. Then, all permutations of pairs of these classes are generated. It means that for p faulty classes there will be $p * (p - 1)$ pairs. This 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 reason: the first class of the pair is then modified so that during balancing (see further) of the pairs, 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 is done so that symbolic classification finds an expression that is worth 0 for nominal samples but also for samples that are member of 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, an expression that 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 $pairsToTry$, the first step is to `balance` samples in the two classes (line 12). This is a pre-processing step that checks which class in the pair has less samples and randomly selects the same number of samples from the class that has the most samples. This is done so that the ensuing 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 pseudo code) always returns an expression candidate, named $foundExpression$, that is the best it found (line 13). However, this expression might be the best found but still not good enough to be considered 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 diagnosis indicator. This is done by taking two consecutive tests.

T1 Check 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, X_{T_1} being a hyper-parameter of the algorithm.

T2 Check that $foundExpression$ predicts correctly $X_{T_2}\%$ of the data used to find it through symbolic classification. This does not include the data discarded for balance purposes. Indeed, symbolic classification might not be able to fit the data and the output expression might be random and not classify the data correctly. This test ensures that $foundExpression$ is classifying correctly. X_{T_2} is a hyper-parameter of the algorithm.

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 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 included in $\mathcal{D}_{n_i}$. 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 that has 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.

4.4 Hyper-parameters

The hyper-parameters described in the previous section are summarized in Table 1, along with the hyper-parameters for symbolic classification.

DT4X	Default Value
purity threshold X_p	0.95
relevance threshold X_r	0.05
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 DT4X hyper-parameters are described in Section 4.3.

The ϵ parameter is described in Section 4.6. ϵ 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 parameter *population size* corresponds to the number of candidate solutions generated at each generation of symbolic classification. The higher it is, the more likely it is that convergence towards a good solution will be fast. However, the larger it is, the more time the training will take.

The *maximum number of generations* is the number of generations beyond which the algorithm will stop even if it

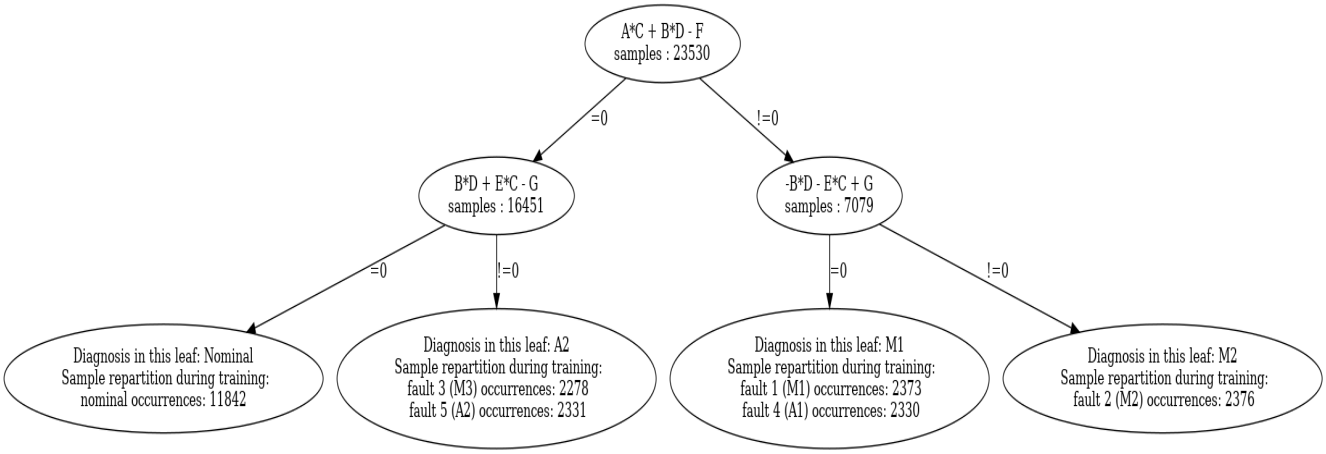

Figure 1: Single Faults Decision Tree

did not find any solution. The bigger it is, the more chances the algorithm has to find the right solution. However, when there is no solution to be found, a higher number might lengthen the time it takes to stop.

The *proportion of samples used* is the proportion of the dataset that will be used to test each candidate solution. It allows a trade-off between computation time and accuracy. Since, in this case, a better accuracy means finding better diagnosis indicators, and since a better diagnosis indicator means a faster prediction, the proportion of samples used should remain high. Indeed, prediction time should have priority over training time. Thus, the whole dataset is used by default.

When computing the fitness of a candidate solution, a penalty is subtracted to its score. This penalty is the *parsimony coefficient* multiplied by the length of the expression of the candidate solution. This favors shorter solutions.

4.5 Prediction with DT4X

Once DT4X finished building the tree, it can be used to predict the class of a sample x, meaning to diagnose the system status when x was measured. Prediction is performed by inputting x in the root node. When x reaches a node n_i, the diagnosis indicator $\mathbf{d}_{n_i}$ of this node is evaluated on x. Similarly to what is done 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 made and it is the label of this leaf node.

4.6 Implementation of DT4X

In DT4X, the transformer for the symbolic classification is customized to fit our problem. Indeed, symbolic classification is here used to find diagnosis indicators. Thus, the custom transformer t used is the following:

$$-\epsilon < y < \epsilon \implies t(y) = 0 \quad (1)$$

$$y \leq -\epsilon \text{ or } y \geq \epsilon \implies t(y) = 1 \quad (2)$$

With ϵ a parameter of DT4X. If nominal data is inputted as class 0 and a faulty scenario data is inputted as class 1 and a function is found that has perfect accuracy on this data, then this function is a diagnosis indicator (in the sense that it is null in nominal cases and not null for this faulty scenario and it only involves observable variables). This fault indicator is sensitive to at least the fault used to find it.

5 Illustration on the Polybox Example

DT4X has been tested on the polybox use case. The polybox is a fictional static system that contains five components: M1, M2, M3, A1 and A2. These components are connected as shown on Figure 2. The M components are multipliers (the output equals the product of the inputs) and the A components are adders (the output equals the addition of the inputs).

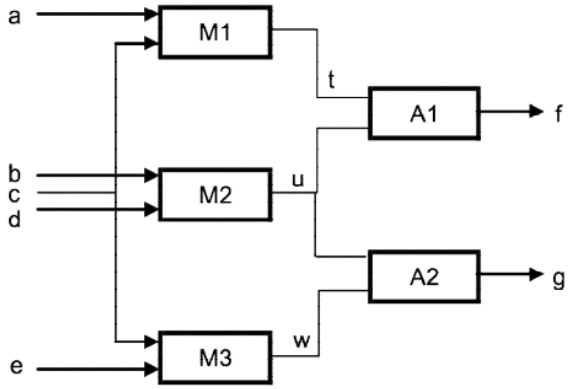

Figure 2: The Polybox

Each component can malfunction, meaning the component does not produce the correct output according to the inputs. Thus, there are five possible faults in this system. For the sake of simplicity, the fault associated with component M1 is called *fault M1* and idem for the other components.

There are seven observable variables in this system: a, b, c, d, e, f and g. The pairs in the dataset are of the shape (x, l) with $x = (a, b, c, d, e, f, g)$ and $l \in C$ with C the set of possible diagnoses. In the context of this study, two experiments have been made. One with single faults only and one with double faults as well as single faults. It can be noted that cases with triple, quadruple and all faults are very similar to the double faults case, and work the same.

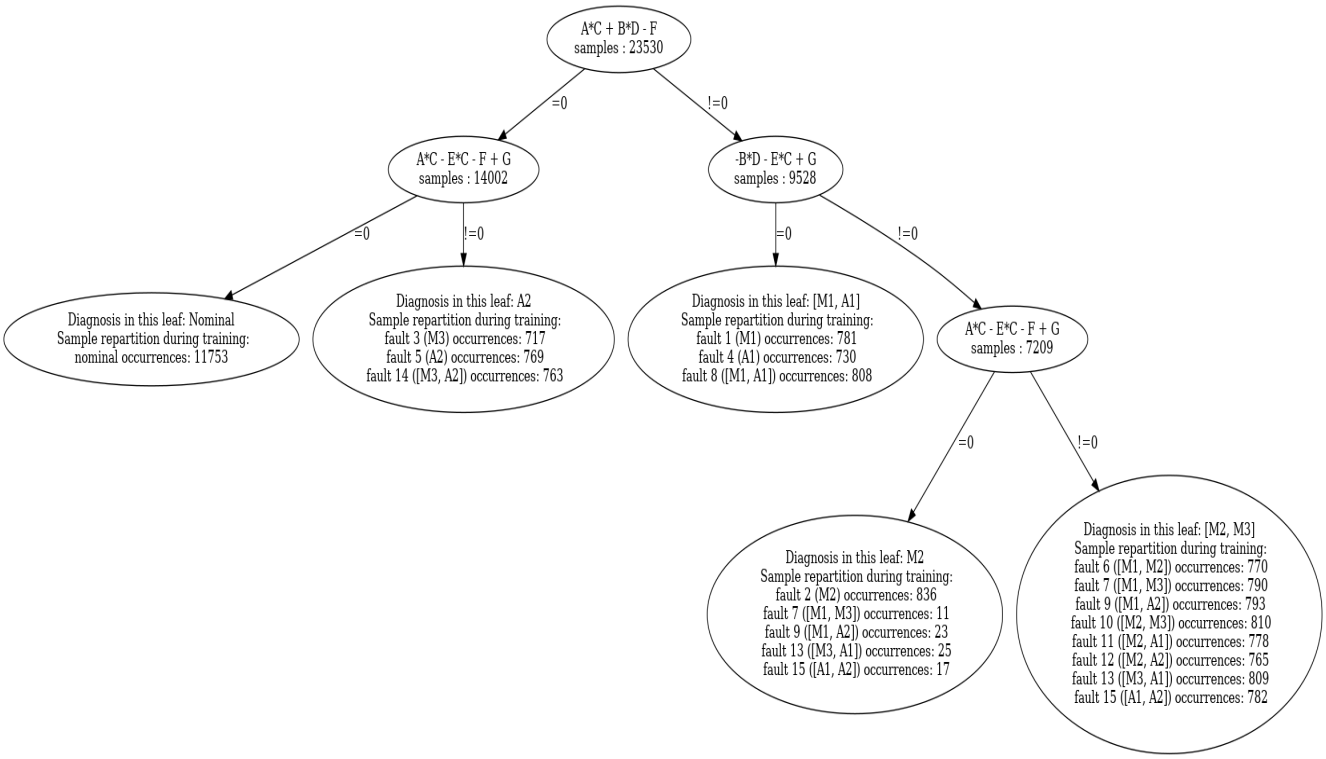

Figure 3: Double Faults Decision Tree

5.1 DT4X results

Single Faults

If only single faults are considered, C = {Nominal, fault M1, fault M2, fault M3, fault A1, fault A2}. The experiment presented here uses a randomly generated dataset of 16807 nominal samples and 16807 faulty samples, each being of one fault type. A fault or a component malfunction is defined as an output from this component that is different from the expected value. Thus, a faulty component outputs the expected value plus a modifier in $\{-3, -2, -1, 1, 2, 3\}$. Then, this dataset is randomly split between a training set with 23530 samples and a testing set with 10084 samples. The training set is injected into DT4X with default hyper-parameters and the output decision tree is shown in Figure 1.

The accuracy of this decision tree on the test set is $80,09\%$.

Double Faults

When considering double faults, C = {Nominal, fault M1, fault M2, fault M3, fault A1, fault A2, fault M1 & fault M2, fault M1 & fault M3, fault M1 & fault A1, fault M1 & fault A2, fault M2 & fault M3, fault M2 & fault A1, fault M2 & fault A2, fault M3 & fault A1, fault M3 & fault A2, fault A1 & fault A2}. This experiment also uses a randomly generated dataset of 16807 nominal samples and 16807 faulty samples, each being of one random fault combination (either single or double fault). A faulty component outputs the expected value plus a random modifier in $[\![-15\ ,\ 15]\!]^*$ but in the case of double faults, the value of the two modifiers are neither the same nor the opposite of each other, in order to avoid fault cancellation. Then, this dataset is randomly split between a training set with 23530 samples and a testing set with 10084 samples. The training set is injected into DT4X with default hyper-parameters and the output decision tree is shown in Figure 3.

The accuracy of this decision tree on the test set is 63.67%.

5.2 Discussion

Classical DT Results

In order to compare DT4X to the more common univariate decision trees, we trained a scikit-learn[1] default decision tree (SklDT) on both single and double fault training datasets.

The accuracy of sklDT on the same single fault test set is 46.85%. Its accuracy on the same double fault test set is 47.33%.

In order to give a better comparison of the two algorithms, we trained a scikit-learn decision tree and a DT4X diagnosis tree for 10 different datasets randomly generated (but still with 23530 samples). The results are presented in Table 2. The time columns correspond to the prediction time for the whole test set (10084 samples) with the same hardware.

The training time for the scikit-learn decision tree is around one second for single faults and five seconds for double faults while DT4X takes around fifteen minutes for single faults and one hour and twenty minutes for double faults, both trained on the same hardware (an AMD Ryzen 9 6900hx with radeon graphics, 16 cores). Overall, a factor of a thousand between the two. In addition, Table 2 shows that the prediction time for scikit-learn decision trees are shorter.

Nevertheless, the univariate decision is not interpretable right away. A set of conditions on variables can be extracted by following the paths to a certain class. In the case of

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

	SklDT		DT4X	
Id	accuracy (%)	time (s)	accuracy (%)	time (s)
1	47.83	0.08	79.93	0.64
2	47.27	0.09	80.30	0.63
3	47.10	0.08	80.00	0.64
4	47.46	0.08	80.14	0.85
5	47.84	0.08	79.52	0.79
6	45.96	0.09	80.08	0.66
7	47.17	0.08	80.59	0.94
8	42.28	0.09	80.16	0.94
9	47.81	0.09	80.66	0.64
10	46.63	0.09	79.72	1.32

Table 2: Comparison of DT4X and scikit-learn decision trees

DT4X, the expressions in the nodes are diagnosis indicators on their own, as it will be explained later.

A comparison with a multivariate decision tree parameterized and trained specifically for the polybox case (rather than the sci-kit learn univariate decision tree) would be more relevant and is a priority among our future works. However, we are confident a better accuracy cannot be reached (see the next section).

Analytical Redundancy Approaches

Model-based diagnosis can be performed using analytical redundancy approaches. They extract fault indicators called *Analytical Redundancy Relations* (ARR) from the model of the system [16].

Definition 3. *An analytical redundancy relation (ARR) is a constraint deduced from the system model which contains only observed variables, and which can therefore be evaluated from any observable sample x. It is noted r = 0, where r is called the residual of the ARR.*

ARRs are used to check the consistency of the observations with respect to the system model. The ARRs are satisfied if the observed system behavior satisfies the model constraints. ARRs can be obtained from the system model by eliminating the unknown variables.

ARRs are relations between observable variables that are sensitive to certain faults, in the sense that they are false in these faulty situations and they are true otherwise.

The ARRs for the polybox are presented in Table 3. An example on how to obtain them is presented in [16]. This representation is called a signature matrix. It indicates in which cases an ARR is verified or not.

	N	M1	M2	M3	A1	A2
$ARR_1 = a*c+b*d-f$	O	X	X	O	X	O
$ARR_2 = e*c+b*d-g$	O	O	X	X	O	X
$ARR_3 = c(a-e)+g-f$	O	X	O	X	X	X

Table 3: Signature Matrix of the Polybox

N means Nominal, M1 means fault M1, and idem for the other faults. An O means that the ARR is verified (is equal to 0) for this case, and a X means it is not.

Let us consider the single fault decision tree in Figure 1. The diagnosis indicator found in the first node corresponds to ARR_1 in Table 3. The samples that went through the left side of the tree are those that verify it. They are those from classes nominal, fault M3 and fault A2. These faults are exactly the ones ARR_1 is insensitive to. The same reasoning can be applied to ARR_2. Actually, the first two lines of the signature matrix can be built from the tree, and vice versa. The first two lines of the signature matrix and the single fault tree are strictly equivalent.

In the double fault tree, the same reasoning can be applied to find the three ARRs. However, in this case, there is no strict equivalence between the tree and the signature matrix. Indeed, the signature matrix allows to build the tree, but not the other way around. For instance, the diagnosis indicator $b*d+e*c-g=0$ is true for all scenarios of {fault M1, fault A1, fault M1 & fault A1} and false for all scenarios of {fault M2, fault M1 & fault M2, fault M1 & fault M3, fault M1 & fault A2, fault M2 & fault M3, fault M2 & fault A1, fault M2 & fault A2, fault M3 & fault A1, fault A1 & fault A2}. However, with the tree alone we can not say whether it is sensitive to the other faults. Though, considering the way it was built, it is expected to be insensitive to the nominal samples. Summarizing, there is more information in the matrix than in the tree and all the information of the tree is contained in the matrix. Nevertheless, the information that is lacking in the tree to reconstruct the signature matrix is information that is not useful to perform full detection and isolation of the faults. The information that the matrix possesses but not the tree is useless to achieve maximal diagnosability.

We indeed conjecture that the tree contains the minimal information for maximal fault diagnosability. This makes sense because analytical redundancy approaches show that the accuracies obtained with single fault and double fault trees are the highest accuracies achievable for this system and this dataset.

Indeed, the signature matrix shows that fault M3 and fault A2 are not isolable. Same for fault M1 and fault A1. Thus, fault M3 samples are predicted as fault A2 and fault A1 samples as fault M1. Considering that the dataset contains 50% of nominal samples, approximately 10% of each fault samples and 2 of those faults are non isolable, the accuracy can not be reliably more than 80% which is what the single fault tree gives according to Section 5.1.

Applying the same reasoning to double fault trees gives a maximum reliable accuracy of 63.33%. This conjecture is to be studied in future work.

In the decision trees presented in Figures 1 and 3, a label is given to leaves that contain multiple classes based on the class that has more samples. However, this is just to be able to perform classification automatically. In an online environment where the system is used to give advice to an operator, or in order to give a relevant diagnosis, the algorithm can output all the classes present in a representative amount in the node reached by the tested sample. This avoids having to attach only one label to a node that contains two classes with pretty much the same amount of samples.

6 Conclusion

The DT4X algorithm builds a multivariate decision tree that performs diagnosis. Decisions in each node are taken based on a diagnosis indicator found through symbolic classification. The obtained decision tree is easily interpretable because each discriminating expression in nodes is an explicit

diagnosis indicator. It has been tested on the well-known polybox system and has been confirmed to output great accuracies. However the question is raised about the possibility to apply this algorithm to more computationally complex systems. Experiments show that the DT4X training step is costly and more complex than a univariate decision tree training. However, its accuracy on a test set is much better and it allows discovering fault indicators otherwise obtainable only through knowledge about the system model.

Future work will improve DT4X by adding some information on the structural model of the system. This is possible by biasing the starting generation of symbolic classification. The next step is testing DT4X on logic circuits and dynamic systems. Interestingly enough, in the case of the polybox, ARRs are found. However, the constraints imply that any diagnosis indicators can be found, not necessarily ARRs. Future research will show whether this is the case on logic circuits and dynamic systems.

References

[1] Khaoula Tidriri, Nizar Chatti, Sylvain Verron, and Teodor Tiplica. 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, 2016.

[2] Yaguo Lei, Bin Yang, Xinwei Jiang, Feng Jia, Naipeng Li, and Asoke K. Nandi. Applications of machine learning to machine fault diagnosis: A review and roadmap. *Mechanical Systems and Signal Processing*, 138:106587, 2020.

[3] Anuja Priyam, Gupta R Abhijeeta, Anju Rathee, and Saurabh Srivastava. Comparative analysis of decision tree classification algorithms. *International Journal of current engineering and technology*, 3(2):334–337, 2013.

[4] Johan de Kleer and James Kurien. Fundamentals of model-based diagnosis. *IFAC Proceedings Volumes*, 36(5):25–36, 2003. 5th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes 2003, Washington DC, 9-11 June 1997.

[5] Belarmino Pulido and Carlos Alonso. Possible conflicts, arrs, and conflicts. In *DX'02 XII International Workshop on Principles of Diagnosis*, 07 2002.

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

[7] Arman Mohammadi, Mattias Krysander, and Daniel E. Jung. Analysis of grey-box neural network-based residuals for consistency-based fault diagnosis. *IFAC-PapersOnLine*, 2022.

[8] Yacine Izza, Alexey Ignatiev, and Joao Marques-Silva. On explaining decision trees, 2020.

[9] Jaime G. Carbonell, Ryszard S. Michalski, and Tom M. Mitchell. 1 - an overview of machine learning. In Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, editors, *Machine Learning*, pages 3–23. Morgan Kaufmann, San Francisco (CA), 1983.

[10] Olcay Taner Yıldız and Ethem Alpaydın. Univariate and multivariate decision trees, 2000.

[11] Zhiwei Gao, Carlo Cecati, and Steven X. Ding. A survey of fault diagnosis and fault-tolerant techniques—part i: Fault diagnosis with model-based and signal-based approaches. *IEEE Transactions on Industrial Electronics*, 62(6):3757–3767, 2015.

[12] Carla E. Brodley and Paul E. Utgoff. Multivariate decision trees. *Machine Learning*, 19(1):45–77, Apr 1995.

[13] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[14] Trevor Stevens. Gplearn. *Github*, 2016. (https://gplearn.readthedocs.io/en/stable/intro.html).

[15] C.M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. p. 209.

[16] M.-O. Cordier, P. Dague, F. Levy, J. Montmain, M. Staroswiecki, and L. Trave-Massuyes. Conflicts versus analytical redundancy relations: a comparative analysis of the model based diagnosis approach from the artificial intelligence and automatic control perspectives. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(5):2163–2177, 2004.