FIREMAN

WP5 Deliverable 5.1
Initial Report on Detection and Prediction Techniques

Project Title:	Framework for the Identification of Rare Events via Machine learning and IoT Networks

Title of Deliverable:	Initial Report on Detection and Prediction Techniques
Status-Version:	Final-1.0
Delivery Date:	31/12/2020

Contributors:	Ioannis T. Christou, Daniel Gutierrez-Rojas, Pedro Nardelli, Pavol Mulinka, Merim Dzaferagic, Irene Macaluso
Reviewers:	John Soldatos, INTRASOFT Intl; Renan Moioli, UFRN (Brazil)
Approved by:	All Partners

Document Revision History

Version	Date	Description
v0.1	13 November 2020	Table of Contents
v0.2	27 November 2020	Main contents added
v0.3	4 December 2020	Additional contents added
v0.4	7 December 2020	Version delivered to the project coordinators
v0.5	8 December 2020	Version delivered for external review
v0.6	22 December 2020	Version updated after external review
v1.0	23 December 2020	Approved form of Deliverable D5.1

Summary

This document summarizes the initial studies about data-driven techniques for rare event detection in industrial settings in the context of WP5. The consortium has focused most attention to study and further develop Quantitative Association Rule Mining algorithms, but has also tested other machine learning and deep learning techniques. The scenarios considered here are related to previous activities carried out in WP2, where the different test cases were defined with their limitations. In addition, the research is closely related to the data fusion research in WP4. The research is performed within the context of the EU CHIST-ERA project FIREMAN (grant CHIST-ERA-17-BDSI-003).

Table of Contents

1 Introduction

1.1 Objective of the document

This deliverable report presents the progress up to now in using advanced Machine Learning and Data Mining techniques for the detection of rare events such as the accumulation beyond a certain safety threshold of sufficient tear-and-wear of machine components that leads to their inevitable breakdown; or simply, of faults within a system or production process that renders it unsafe for continuing its operation; and subsequent prediction techniques that can predict the Remaining Useful Life (RUL) of a machine/component combination or the time of the next appearance of a fault. In all the above-mentioned cases, the eventual break-down of a machine or component is the rare event of interest. Depending on the particular scenario under study, these events may occur once or twice per year, but may also occur even less frequently. Their importance is usually reversely proportional to their frequency of occurrence.

In the context of FIREMAN, we have chosen to study rule-based classifier systems as well as deep neural networks as tools for predicting various indicators of the health of a machine/component or entire industrial process. Our decision to study deep neural networks stems mainly from the phenomenal success of neural networks of many layers in a large variety of tasks including predictive maintenance: a recurrent neural network won the IEEE PHM (Prognostics and System Health Management) competition in 2008. Similarly, the success of rule-based classifier ensembles in research projects related to Predictive Maintenance (PdM) –see for example https://www.prophesy.eu– combined with the strong explainability of rule-based systems, as well as the fact that rule-based systems work by deriving minority class rules before deriving majority class rules, compelled us to study such algorithms as well.

1.2 Structure of the document

The document is structured as follows: in section 2 we provide a short description of the real-world data that our industrial partner SEAT will make available to the consortium. In section 3 we describe pre-existing research on Quantitative Association Rule Mining (QARM), and in particular the work already done that has lead to a sophisticated implementation of the QARM algorithm (QARMA). In section 4 we describe the work done within the context of the FIREMAN project utilizing and extending QARMA according to the needs of the project. In section 5 we briefly outline other ML techniques we intend to explore in the project. And in section 6 we describe the various Deep Learning approaches in particular that we have already started implementing in the project and will continue to do so throughout its duration. Then, in section 7 we describe computational results we have already obtained from several of our implementations. Finally, in Section 8, we present our conclusions and future directions to take in the duration of the project.

2 SEAT datasets

SEAT provides datasets related to two predictive maintenance use cases derived from its main factories: SEAT Martorell, SEAT Barcelona, and SEAT Components.

The first use case focuses on the early detection of failures in mechanical components (bearing) of the drive chain in paint shop, causing axial displacement that results in significant production stops and corresponding loss of bodies. Available datasets integrate miniaturized data (i.e., 1440 samples of data per day and variable), stored minute by minute about engine group binary signals, tensor, security, position sensors (axial monitoring), etc. The datasets include identified failures for previous years.

The second use case focuses on the prediction of failures in spindles of Computer Numerical Control (CNC) machines of the machining center, generating defects in components of the gearboxes produced in the SEAT components plan. Available datasets are composed of over 50 variables, such as axis positions, engine energy consumption, pressure, temperature, fluids levels, preventive control points, failures, quality reports, etc. The datasets provide no identified failures for previous years.

In the context of WP5/WP6 research activities, data analysis will include features variance, distribution, collinearity, missing values, same values, label imbalance, etc. Primary focus will be placed on the first use case (drive chain dataset). Depending on the preliminary analysis of the datasets and their features, we aim to (i) apply and evaluate machine learning approaches developed within the FIREMAN project and (ii) discuss/propose data acquisition guidelines, e.g., installation of additional sensors and other hardware and software (HW/SW) elements installation for failure detection.

3 Pre-existing Research on QARMA

QARMA is a family of algorithms for extracting all (or, depending on user inputs, an important subset of) valid non-dominated quantitative association rules that hold in a dataset, that can then be used for further data analysis such as deriving rule-based classifier ensembles or as explainers of classification results of other black-box classifiers. A detailed description of the basic algorithm is given in [1], [2]. Some uses of the basic QARMA algorithm are described in [3] and [4].

In this section, we describe the input formats that QARMA (Quantitative Association Rule Mining Algorithm) expects, the outputs it produces, and some graphical user interface (GUI) based applications that have been developed to analyze them.

3.1 Input Datasets of QARMA

QARMA works on datasets that can be viewed as histories of "user" transactions, consisting of "purchases" of "items" that have various "attributes". Each such item purchase by a user, defines a transaction, for which the following data are available:

1. A user-id of the "user" who makes the "purchase"
2. An item-id of the "purchased" "item" by the "user"
3. Attribute-value pairs of the form (attr-id, value) for some (at least one) of the "item"'s attributes.

As an easy example, in a on-line video-club, subscribers (the users) may rent movies (the items) for the current price of the movie, so that "price" is the first attribute of every item; then, after seeing the movie, the subscriber may give it a rating, so that "rating" becomes a (second and optional) attribute of each item.

As a second example, in a medical domain example, the "users" may simply be instances of values for various blood tests, and a "class" value may indicate whether the given tests indicate the presence of a disease or not. In this case, the "items" are the various blood tests (what is commonly referred to as "attributes" in standard supervised machine learning literature), and each "item" has a single "attribute", that is its value. The class value is yet another item, with a single attribute, its value that can be zero (no disease) or one (disease present).

3.2 QARMA Main Functionalities

QARMA is a system for extracting all valid, non-dominated quantitative association rules that can be extracted from a dataset that define multiple items and restrictions on zero or more of their attributes' values in the antecedents of the rule, and a single item and a restriction for a single attribute of that item. By a "restriction" of an item-attribute, we mean the requirement that the item-attribute's value is at least as large as the value specified in the restriction in the rule.

The system considers valid any rule that meets minimum required support and confidence levels. The system considers a rule r1 to be dominated by another rule r2 when:

1. the consequent items of the rules are the same, as well as the consequent item's restricted attribute is the same,
2. the antecedent items in r1 are a (non-strict) super-set of the antecedent items of r2,
3. the restricted threshold value for r2's consequent item attribute specified is at least as big as the corresponding one specified for r1,
4. all the restricted attributes of antecedent item attributes in r2 also appear restricted in r1, and the values in r1 are at least as big as those in r2,
5. the support and confidence levels of r1 are at most as big as those of r2.

All restrictions specified for an item in a rule must be satisfied simultaneously by at least one transaction in a user's history; when a user history contains transactions for each item specified in a rule, and for each item in the rule, there exists a transaction in the user's history satisfying all the specified restrictions for that item, the user history satisfies the rule. The above only makes sense when the user is allowed to make multiple purchases of the same item, and it will be more easily understood with an example. In online video-clubs, it is commonly observed that many customers rent the same movie multiple times (this is particularly prevalent for children's movies). Consider the rule PRICE_PAID("Usual Suspects")>=2 AND IN_HIGH_DEF("Usual Suspects")>=1 AND PRICE_PAID("Fargo")>=0 PRICE_PAID("Pulp Fiction")>=1. This rule is to be read as follows: if a user purchases "Usual Suspects" for at least $2 to see it in HD format, and if they also purchase the movie "Fargo", then this user is willing to pay at least $1 for "Pulp Fiction". The rule would not be satisfied by a user who has rented "Fargo", and who once rented "Usual Suspects" for $2 in SD, and then at another time, in a special offer, rented "Usual Suspects" again, in HD this time, for $1.

QARMA may be asked to produce only, those valid non-dominated rules that are "widest". A rule r2 is "wider" than r1 iff the pair of these rules satisfies all conditions 1-5 with the possible exception of the confidence level relation between r1 and r2. QARMA may also be asked to produce all rules with up to a certain size of rule length; it may also be asked to consider the consequent's item attribute restriction to be specifying an equality ('=') rather than inequality ('>').

It may also be asked to produce only rules with a given item-id in the consequent. This is particularly useful in a standard "supervised learning" context, where we wish to produce rules that classify instances, so we want only the "class attribute" to be the consequent item (in combination with the previous listed capability). And, in the case when some attributes of some items take on too many distinct values in the dataset (e.g. the blood test values in the UCI ilpd dataset), it can be asked to consider only a "reasonable" subset of these values when creating rules containing restrictions on such item attributes. There are more short-cuts and heuristics that QARMA can be asked to employ, but they are not in scope for this document to describe.

3.3 QARMA Input Formats

There are two major data formats in which data can be formatted for loading onto the QARMA database. For both of these formats, there is an implicit assumption that each "item"

has exactly the same attributes as any other item in the dataset, and that there always exists one "major" common attribute, for which, in every user transaction, this attribute has a value in the transaction.

1st INPUT FORMAT: Extended MovieLens style format

The first input format is a text-based (ASCII) extension of the MovieLens CSV format. In this format, the first line of the file has the following format:

<attr1_name>::<attr2_name>::. . .::<attrn_name>

Where <attrk_name> is the name of the k-th attribute of each item in the dataset, besides the "main" common attribute of each item. If there are no "extra" attributes other than the "main" common attribute, this first "header" line can be omitted. Notice the use of the double column ("::") as separator.

The rest of the file contains lines of the form:
<user_id>::<item_id>::<main_attribute_value>::<attr1_val>::<attr2_val>. . . <attrn_val>
Where user_id is an positive long number uniquely identifying a user, item_id is a positive long number uniquely identifying the particular item the user has "purchased", main_attribute_value is a real number representing the value of the main attribute of the item, and attrk_val is a real number representing the value of the k-th attribute (whose name is <attrk_name> in the header), that can be the character '?' to indicate that the value is "N/A" or "unknown". Notice that double-column ("::") is used again as values separator.
An example of such an input file is the following (only the first 5 lines are shown for brevity):
amount_played::amount_won::ngpr
1000005332::1::0.5::1000000::0::1
1000005664::1::1::200000::0::1
1000005664::1::0.15::200000::0::1
1000005664::1::2::25000::0::1
1000008703::1::0.04::750000::400000::2
Notice that in the above dataset, the user with id "1000005664" has purchased item with id "1" 3 times, with different values for the standard "main_attribute" attribute, as well as for the "amount_played", "amount_won" and "ngpr" attributes. This particular feature is not commonly found in other algorithms in the ML/DM literature.

2nd INPUT FORMAT: CSV Format

The 2nd format is a text-based input format in which every line contains the entire history of a user. Rows therefore represent users histories, and columns represent the items in the dataset. Therefore, the j-th column of the i-th row is a value that represents the major value that the i-th user "paid" for the j-th item.

The 1st row of the file is a header with the names of each item in the dataset:
<item1_name>,<item2_name>,. . . <itemn_name>
Where <itemk_name> is the name of the k-th item in the dataset. Notice the use of the comma (',') as separator.

The rows afterwards have the following format:
<item1_value>,<item2_value>,. . . ,<itemn_value>

Where <itemk_value> is the value that the i-th user "paid" for the k-th item, and is '?' if the user did NOT "purchase" the k-th item. Again, comma (',') is used as values separator.

An example file excerpt containing data from the UCI ilpd (Indian Liver Patient Database) dataset is the following:

$$age, gender, TB, DB, AAP, Sgpt, Sgot, TP, ALB, AGRatio, sick$$

$$65, 0, 0.7, 0.1, 187, 16, 18, 6.8, 3.3, 0.9, 1$$

$$62, 1, 10.9, 5.5, 699, 64, 100, 7.5, 3.2, 0.74, 1$$

$$62, 1, 7.3, 4.1, 490, 60, 68, 7, 3.3, 0.89, 1$$

$$58, 1, 1, 0.4, 182, 14, 20, 6.8, 3.4, 1, 1$$

In the above file excerpt, the top row is the header, the 2^{nd} row represents the data for the 1^{st} user, the 3^{rd} row represents data of the 2^{nd} user and so on.

3.4 QARMA Outputs

QARMA writes ALL the rules it produces in its database (a MySQL RDBMS database).

The schema where these rules are stored is shown in Figure 1. The tables that are populated with the rules found are the tables:

1. associationrule
2. associationruleitem
3. associationruleitemattrs

The other tables shown in the figure must have been populated (as well as some other tables not shown in the figure) before each run.

QARMA also writes its rules in a plain text file (whose name is provided as a parameter when invoked from the command line), with 1 line for each rule found, printing the rule first, then the support, confidence, lift, and conviction metrics of the rule on the dataset, as in the following example:

...
[MovieLensTitle-110.p >= 1.0] ∧ [MovieLensTitle-457.p >= -Infinity] ∧ [MovieLensTitle-480.p >= -Infinity] -> [MovieLensTitle-1580.p >= 1.0] ...
[MovieLensTitle-1097.p >= 1.0]&[MovieLensTitle-1097.price_negated >= -4.0] ∧ [MovieLensTitle-1270.p >= -Infinity] -> [MovieLensTitle-260.p >= 3.0] ...
[MovieLensTitle-1097.p >= 1.0] ∧ [MovieLensTitle-1270.p >= -Infinity] -> [MovieLensTitle-260.p >= 1.0] ...
...

QARMA also outputs in the console in which it is invoked the total dataset coverage (in terms of the percentage of users covered by the rules found).

In the above example, the dataset (it's the MovieLens ml-1m dataset) contains two attributes, the standard "price" attribute (which is the number of stars each user has rated a movie), plus the negation of the "price" attribute (i.e. minus the number of stars that user rated the movie). This means that the 2^{nd} rule shown above reads as follows:
RATINGS(MovieLensTitle-1097) ∈[1,4] AND RATINGS(MovieLensTitle-1270)>-∞ →RATINGS(MovieLensTitle-260)≥3.

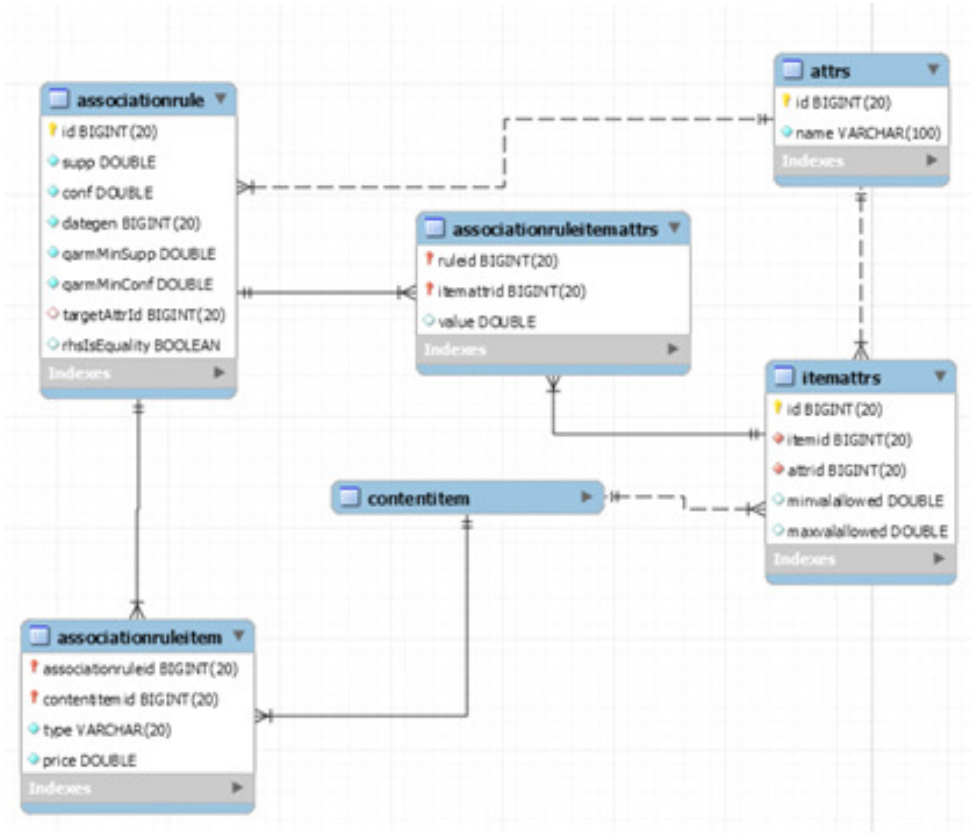

Figure 1: QARMA RDBMS tables for storing produced rules

3.5 QARMA UIs

QARMA comes with a set of 2 GUIs that allow to visualize various aspects of the produced rules. The first GUI allows the user to see all the rules in the DB and upon selection of a rule, to see the "user histories" (data instances) that satisfy the antecedents of the selected rule, along with the value of the target item/attribute. The GUI is shown in Figure 2.

The second GUI allows not only to inspect and rule, but to modify the rule on-the-fly so as to see how the rule's support and confidence metrics change as the intervals in which the antecedents are required to be in change. Figure 3 is a screen-shot of this tool.

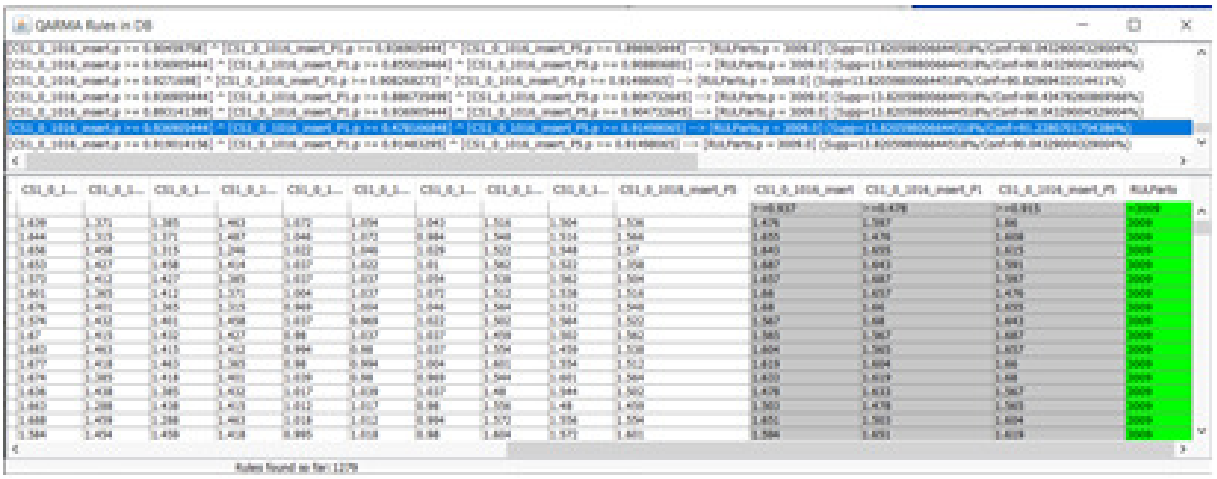

Figure 2: QARMA Rule Visualization Showing All Instances in DB Matching Rule Antecedents. The column with the green background is the target variable.

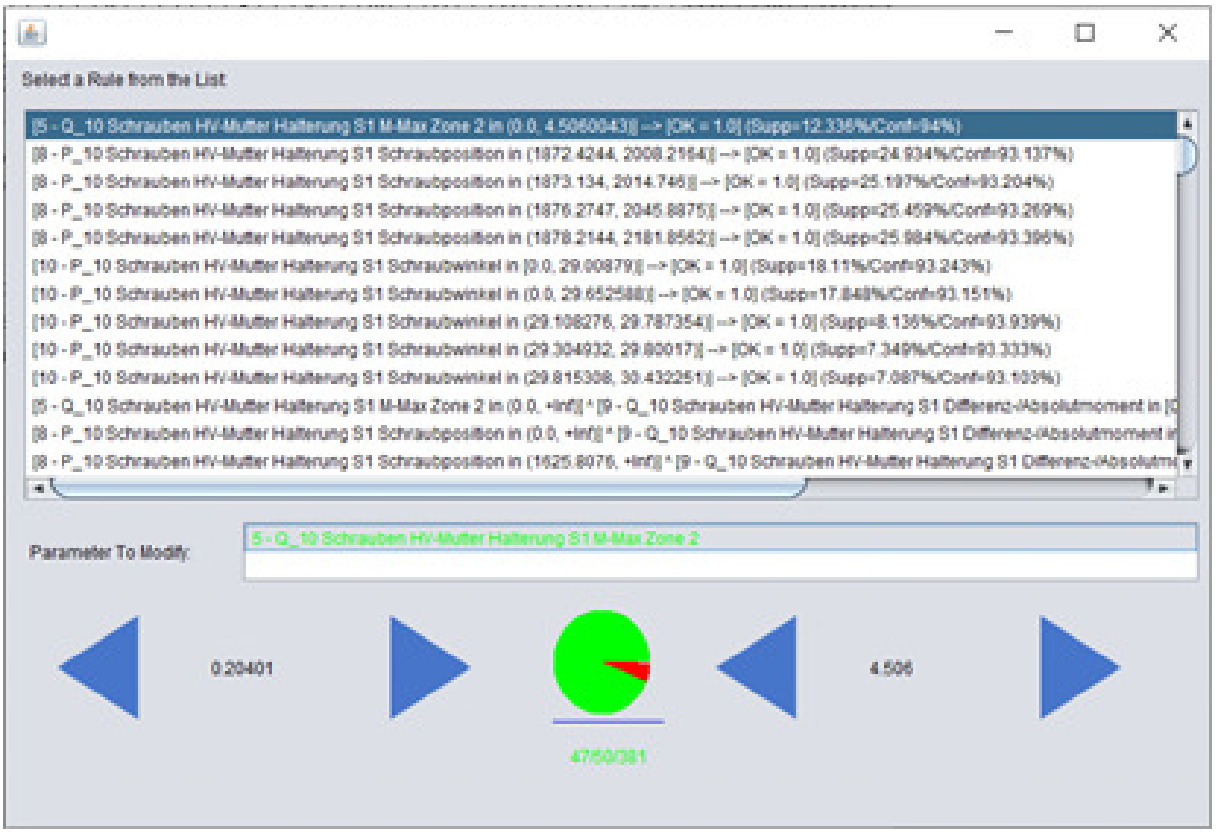

Figure 3: QARMA QuantRule Inspection and Modification Tool: The pie-chart in the middle of the bottom panel shows the confidence of the rule, the straight blue line immediately below shows the selected item's position with respect to the extreme values the item/attribute takes on the training dataset, and the numbers in green below show respectively how many data instances in the dataset satisfy both the antecedents and the consequent of the rule, how many satisfy only the antecedents, and how many instances are the database in total.

4 Progress Report on Quantitive Assosiation Rule Mining for Rare Event Detection

4.1 R4RE

Within FIREMAN, the QARMA algorithmic framework has been expanded in several ways to detect rare events in Industry 4.0 settings. The first direction was to expand the basic algorithm so that it can do early pruning of rules that may pass the support and interestingness metrics required, but only due to chance. Pruning of rules as they get more complex is a well-known technique in rule extraction systems such as SLIPPER, RIPPER-k etc. but in the context of association rule mining, the technique is not often used. In [2] we describe an algorithm that uses the basic idea that some rules that appear to have strong confidence on a training set but barely pass the support threshold, might not maintain their confidence level on an unseen test set, to early prune such possibly poor-performing rules.

4.2 Extracting Minimum Length Variable Sets Explaining Data

A second direction that we have expanded QARMA within the FIREMAN project, is in the direction of "explainability". For dense datasets, it is a frequent phenomenon that QARMA/R4RE may produce thousands of valid rules that hold on the dataset, and are non-dominated. The produced rules contain antecedents that are combinations of all the "items" (ie features) in the dataset, unless the QARMA run that produced those rules was instructed to use only certain items from the entire set of items specified in the dataset. However, in most cases, a very small subset of the entire set of rules produced is enough to "cover" or "explain" most of the data instances that the entire produced rule-set covers. Even more, a small subset of rules that employs in its antecedents a very small subset of the entire set of items (features) that exist in the dataset, is usually enough to "cover" most of the data instances covered by the entire rule-set.

Even though from an algorithmic point of view, the resulting rule-set size may not be of great interest, in practical terms, it is some times helpful to have only a few rules to work with. In one practical scenario, one may seek to derive combinations of production process parameter settings that lead to production runs that lead to few or none defective products. In such cases, the shop-floor engineers would certainly prefer having to analyze only a few combinations (described as rules' antecedent conditions) as opposed to having to browse through many such possible combinations. It is for such scenarios mostly, that we have taken this second direction in FIREMAN.

We have developed a highly parallel/distributed algorithm that can find such a small subset of the QARMA produced rule-set very efficiently, in a fraction of the time that any Open-Source Mathematical Programming solver takes to solve the corresponding Integer Programming formulation. In the following, we first formulate the problem at hand in a rigorous mathematical sense, and then describe a heuristic algorithm for solving it. One of the advantages of our algorithm is that it can be turned into an exact algorithm for finding the solution sought by simply specifying appropriate values for the user-defined input parameters. The second great advantage is its highly efficient use of any and all available computational resources (ie CPU

cores).

In order to fully understand the problem, we first model it as a mathematical optimization problem, then we show how to solve it with highly efficient parallel hybrid Breadth-First/Beam Search heuristic.

We are given a finite set of m variables $V = \{v_1, v_2, ..., v_m\}$ that correspond to the features (attributes) in a finite dataset $D = \{i_1, i_2, ..., i_n\}$. We are also given a finite set of rules $R = \{r_1, r_2, ..., r_k\}$ each of which contains a usually small subset $V(r_j)$ of the variables in V and "explains" or "covers" a subset $I(r_j)$ of the instances in D. By "explaining" an instance i we mean that the rule r_j is satisfied by that instance, i.e. both the rule's antecedent and consequent conditions specified are satisfied by the instance's values for its features.

The objective of the rules-version of the "MinExplainSet" problem is to find a set R_m of rules in R such that its size is less than or equal to a user-defined threshold S and has the property that the instances in D satisfied by at least one of the rules in R_m is no less than a user-defined fraction a of the total number of instances in D satisfied by the entire set R.

To formulate the "MinExplainSet" as an Integer Programming problem, consider the matrix $A = [a_{pq}]_{n \times k}$ where

$$a_{pq} = \begin{cases} 1 \; if \; i_p \in I(r_q) \\ 0 \; else \end{cases}$$

Let $s_j = A_{j.}x$ where $x = (x_1, x_2, ..., x_k)^T$ is a binary vector in $0, 1^k$ indicating if $r_j, j = 1, 2, , , , k$ is in the required solution or not. The variable s_j indicates the number of times instance is "explained" by the set of rules contained in x. Let $b_j = ||A_{j.}||_1$ indicate the total number of rules that "explain" instance $i_j \in D$.

Our problem now becomes:

$$\min_{x,y} 0^T x$$

$$s.t. \begin{cases} \sum_{i=1}^{k} x_i \leq S & (1) \\ \sum_{i=1}^{n} y_i \geq a \left| \bigcup_{r \in R} I(r) \right| & (2) \\ s_i \leq b_i y_i \; \forall i = 1, 2, \ldots, n & (3) \\ s_i \geq y_i \;\; \forall i = 1, 2, \ldots, n & (4) \\ x_i \in \{0, 1\} \;\; \forall i = 1, 2, \ldots, \; k, y_i \in \{0, 1\} \;\; \forall i = 1, 2, \ldots, \; n & (5) \end{cases}$$

The problem variables x, y are both binary vectors. The x variables indicate which rules enter the solution. the y variables indicate if an data instance is covered by the solution or not. The objective function is zero, as we only care to obtain a solution that satisfies the constraints. The 1st constraint specifies that the solution must contain at most S rules. The second constraint specifies that the total number of instances covered by the solution must be at least a times the total number of instances covered by the entire rule-set found. Constraints (3-4) force the variable y_j to be set to 1 if and only if the instance i_j is covered by the solution. Indeed, if $y_j = 1$ then $s_j = A_{j.}x \geq 1$ by constraint (4) so the instance i_j is covered by at least one rule in the solution x, while constraint (3) is still satisfied; and vice-versa, if $y_j = 0$ then by constraint (3) $s_j = 0$ so that the instance i_j is covered zero times by the rules in the solution. The constraints (5) simply indicate the binary nature of all decision variables.

The above problem—with an objective function that is zero everywhere—sounds simple enough for a modern MIP solver to solve, but tests with the current fastest Open Source solver SCIP (version 7) on problems generated from the datasets described in the computational results section show that often this is not the case. Next, we describe an algorithm called MERSFinder for solving the above problem. The algorithm is implemented in Java in the Open-Source library popt4jlib (https://github.com/ioannischristou/popt4jlib).

Algorithm $MERSFinder$

INPUTS: 2-dimensional Boolean array $R2I$ indicating if rule r covers data instance i, 1-dimensional Boolean array $rules$ (usually initially empty) containing a set of rules that must exist in the solution, $c\_m \in \mathbb{N}$ the maximum number of rules allowed in the solution, $J > 0$ the minimum number of instances that the solution must cover, initially empty FIFO queue Q maintaining sets of candidate solutions, initially empty set G of feasible solutions.
$d_m \in \mathbb{N}$ the minimum depth required before Breadth-First search switches to greedy, $K \in \mathbb{N}$ the maximum number of rules to consider when expanding the current solution in Breadth-First search mode.
OUTPUTS: 1-dimensional Boolean array containing the solution, or NULL if no such solution could be found.
1. **let** R be the total number of rules, D be the total number of instances.
2. **set** $s = getSatInsts(rules)$
3. **if** $|rules| > c_m$ **then return** NULL **else if** $|rules| > d_m$ **then set** $K = 1$
4. **add** $rules$ **to** Q
5. **set** $H = minHeap()$
6. **while** $Q \neq \emptyset$ **do**:
6.1. **pop** the top rule-set **from** Q **into** the variable cur
6.2. **if** $|cur| > c_m$ **then continue** // current solution infeasible
6.3. **set** $cs = |getSatInsts(cur)|$
6.4. **if** $cs > c_m \wedge\ |cur| > d_m$ **then**
6.4.1. **add** cur **to** G
6.4.2. **return** $getBest(G)$
6.5. **end if**
6.6. **set** $cur_m = cur \cup_{i=\max(r \in cur)+1}^{R} \{i\}$ // consider adding all rules to current solution
6.7. **if** $|getSatInsts(cur_m)| < c_m$**then continue** // short-cut if solution leads nowhere
6.8. **reset** H
6.9. **for** $i = \max(r \in cur)\ + 1 \ldots\ R$ **do**:
6.9.1. **add** $(i, getCost\,(i, cur))$ **to** H
6.10. **end for**
6.11. **for** $i = 1$ **up to** K **do**:
6.11.1. **pop** the top pair (j, c) **from** H
6.11.2. **if** $|cur_m| > d_m$ **then run** Algorithm $MERSFinder(R2I,\ cur \cup \{j\},\ c_m,\ J,\ Q,\ G,\ d_m,\ K)$ **and add** the result to G.
6.11.3. **else add** $cur \cup \{j\}$ **to** Q.
6.12. **end for** // i
7. **end while**
8. **return** $getBest(G)$

In the above algorithm, the function $getSatInsts(rules)$ returns all the instances covered by at least one rule in the argument set; the function $getCost(r, S)$ returns an estimate of the "cost" of adding the rule r to the solution S that is directly proportional to the number of rules in the new solution and reversely proportional to the number of instances the new solution covers; and the notation $|s|$ refers to the cardinality of set s. Finally, the notation $max(r \in R)$ indicates the maximum rule id in set R.

The algorithm above is implemented in package $popt4jlib.MinExplainSet$ in class $BottomUpMERSSolver$ and is a recursive serial algorithm. However, it is easy to parallelize, utilizing a thread-pool that can accept batches of tasks to execute asynchronously. One such highly efficient parallel version of the above algorithm is implemented in class $BottomUpMERSSolver2MT$ within the same package, in the popt4jlib Open-Source library. The Breadth-First part of the search can be easily parallelized and assigned to different cores to run in parallel, in an asynchronous manner: tasks run in parallel without the caller who submits them to have to wait for their completion. In the beginning of the algorithm, a shared condition counter is incremented to one, and a single basic task corresponding to the empty set is submitted to the thread-pool. The basic thread that started the computation awaits for the condition counter to reach the value zero, which indicates that all computations have finished. Each task corresponding to a current rule-set S submitted is executed by one thread in the thread pool. If the task corresponds to an initial solution r having cardinality less than d_m, then each of the top K rules with ids above $max(r \in S)$ are successively appended to S and submitted as new tasks to the thread-pool. Otherwise, the current solution is expanded greedily according to the $getCost()$ function to either produce a feasible solution or discover that the expanded solution is not feasible and return NULL instead. Before a task is submitted to the thread-pool, the shared counter is (atomically) incremented by one, and right before a task ends its execution in the thread-pool, it decrements (atomically) the shared counter by one. This ensures that the main thread of computation will only be signaled and return from the await call where it is waiting when all tasks that must be processed have executed.

Initial results on the datasets described in section 7 show that the $MERSFinder$ algorithm helps speed up the resulting run-time rule-ensemble, without any important performance hits in terms of classification accuracy. A full set of results regarding this approach will be published in a paper that is still under active development.

Besides the rules-version of the minimum explaining set problem where we seek a minimal number of rules that cover most of the instances in the training set, within WP2 of the FIREMAN project, in deliverable D2.3 we described the variables version of a variant of that problem, that serves as a dimensionality reduction technique. The variable-version of the algorithm seeks to identify a minimal number of variables that are the only ones used in a set of rules (that may be a lot of them) that cover most of the instances in the training set. The algorithm is shown next for completeness and comparison purposes with the rules-version of the problem. One implementation is open-sourced, also available in the package $popt4jlib.MinExplainSet$ in class $BottomUpMESSolver$ in the library (https://github.com/ioannischristou/popt4jlib). The pseudo-code is given below, in Algorithm Minimum Explaining Variables, where n_{min}, K are user-defined values for controlling the search.

As shown in the algorithm description, the function $getBest(sols)$ accepts as input a set of sets of variables and returns the one among them that has maximum training set coverage. The

Algorithm 1: Min. Explaining Variables Algorithm

```
1  sols ← ∅;
2  Q ← {∅};
3  while |Q| > 0 do
4      vars ← Q.pop();
5      vars.add(targetVar);
6      satInsts ← getSatisfyingInstances(vars);
7      if |satInsts| > n_min then
8          sols.add(vars);
9          return getBest(sols);
10     vars.remove(targetVar);
11     candVars ← getBestOfRemainingVars(vars, K);
12     while |candVars| > 0 do
13         v ← candVars.pop();
14         if index(v) ≤ max_{a∈vars} index(a) then
15             continue;
16         Q.add(vars ∪ {v})
17 return getBest(sols)
```

function $getSatisfyingInstances(vars)$ receives as input a set of variables and returns the set of training instances that satisfy at least one of the rules whose preconditions and postcondition are a subset of $vars$. Finally, the function $getBestOfRemainingVars(vars, K)$ returns the top K variables x that are *not* in the set $vars$ when the variables are sorted in ascending order of the quantity $(|var| + 1)/(getSatisfyingInstances(vars \cup \{x\}) + 1)$ and represents a greedy evaluation of the current cost of adding a variable to a candidate set.

5 Other Machine Learning Techniques for Fault Detection, Diagnosis and Prediction

In FIREMAN, we shall investigate various other machine learning techniques in the context of WP5/WP6 data analysis. In particular:

- Batch machine learning: we will apply several well-known classification and clustering algorithms from scikit and ELKI, tune their parameters and evaluate, analyze their performance. Our main focus will be to create basis for analysis of the stream-based, QARMA and deep learning techniques performance.
- Stream-based machine learning: we will emulate the real-world scenario of CPS industrial environment by transforming the TEP and SEAT datasets to simulated data streams. We will apply the state-of-the-art techniques from well known stream-based machine learning libraries scikit-multiflow and MOA and analyse their performance when dealing with evolving data streams, i.e. streams which characteristics change in time.

In the next section, we will present initial results on another techniques based on Deep Learning.

6 Deep Learning Techniques

6.1 Data Imputation

In industrial use cases involving sensor connectivity, missing data arise inadvertently, e.g., due to connectivity impairments, hardware failures, security attacks, etc., rendering the statistical analysis a complex task. Missing-data imputation constitutes the process of replacing missing data with substituted values.

In the context of FIREMAN, we aim to summarize state-of-the-art guidelines for deep learning topology design and hyper parameter tuning in data imputation problems. We will implement a general approach in PyTorch framework and extend it to two implementations of (i) Variational Autoencoders and (ii) Generative Adversarial Imputation Networks (GAINs). We will define a set of scenarios reflecting possible real-world issues when the measurements are not being received or the sensors stop working. We will define a base classifier and compare its results on (i) the original dataset and (ii) imputed dataset in each scenario. The performance will be compared to the elementary imputation techniques, e.g., rolling average/min/max/, etc.

6.1.1 Variational Autoencoders

Regarding the applicability of variational autoencoders in data imputation tasks, we aim to extend the work of the authors in [5] by (i) proposing guidelines on deep net design, (ii) thorough comparison to other available methods and (iii) application of the method on the real-world datasets provided by SEAT.

6.1.2 Generative Adversarial Imputation Networks (GAINs)

Since the overall goal is to detect anomalies in the presence of missing data, we combine the imputation mechanism with a classifier trained to detect anomalies. In the current implementation this classifier is a neural network that receives as input the current sensor measurements and the previously reported measurements (l lags). This neural network is trained without missing data using the Extended Tennessee Eastman dataset training files (https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/6C3JR1).

The imputation mechanism we investigate is based on the Generative Adversarial Networks framework, and in particular on the work in [6]. We extend this approach to address both temporary and persistent missing data and we evaluate its performance based on the impact on the anomaly detection classifier rather than using the RMSE. Preliminary results in Figure 4 show that the GAIN performs well in correspondence to a missing rate of 10%.

We also examine the impact of persistent single sensor failures on the anomaly detection when replacing the missing data with the average computed over the training data in the absence of anomalies (fault free training data). Results in Figure 5 show that in most cases a simple average-based imputation is enough to guarantee a performance greater than 80% (cells in dark green in figure), and that in $\sim 74\%$ of the cases the probability of detection is higher than 95%.

We will extend the GAIN based approach to persistent failures of multiple sensors.

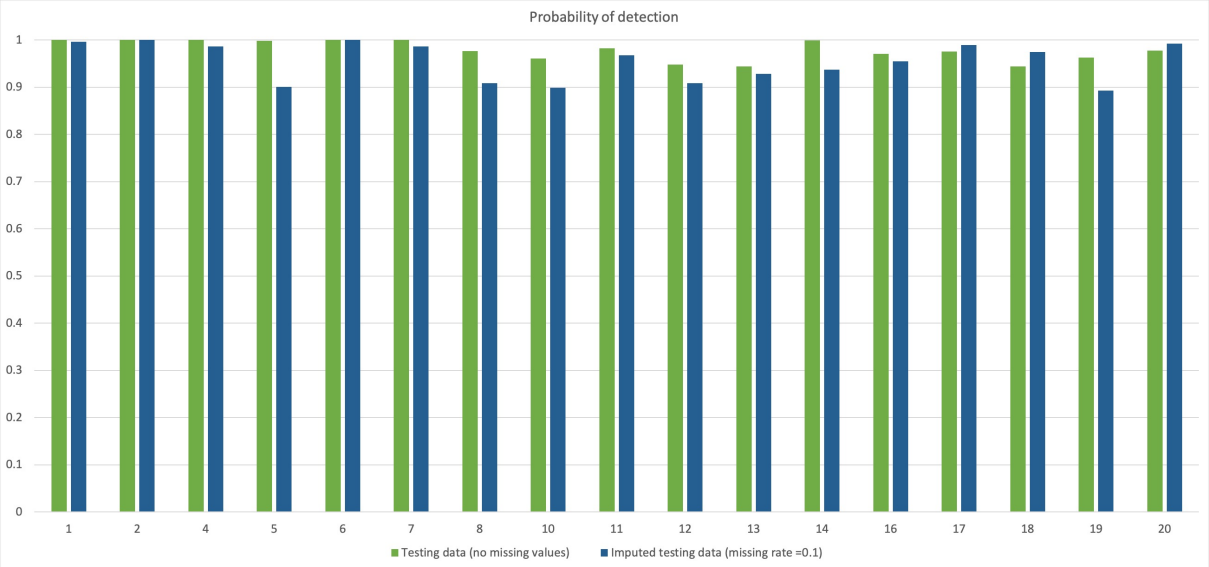

Figure 4: Probability of detection for different faults in correspondence to the testing data. Missing data are replaced by values generated using GAIN.

Sensor	Fault	1.00	2.00	4.00	5.00	6.00	7.00	8.00	10.00	11.00	12.00	13.00	14.00	16.00	17.00	18.00	19.00	20.00
0		0.03	1.00	1.00	0.98	0.98	0.97	0.96	0.96	0.98	0.93	0.94	1.00	0.97	0.98	0.92	0.96	0.98
1		1.00	1.00	1.00	1.00	1.00	1.00	0.97	0.96	0.98	0.95	0.94	1.00	0.97	0.98	0.94	0.96	0.98
2		1.00	1.00	1.00	0.99	1.00	1.00	0.97	0.96	0.98	0.94	0.94	1.00	0.97	0.98	0.94	0.96	0.98
3		0.96	1.00	1.00	0.97	1.00	0.96	0.93	0.95	0.98	0.84	0.94	1.00	0.97	0.98	0.87	0.96	0.98
4		1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.98	0.95	0.95	1.00	0.97	0.98	0.94	0.92	0.98
5		1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.98	0.95	0.94	1.00	0.97	0.98	0.94	0.96	0.98
6		0.98	1.00	1.00	0.95	1.00	0.94	0.96	0.95	0.98	0.88	0.90	1.00	0.97	0.98	0.89	0.95	0.98
7		1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.98	0.94	0.94	1.00	0.97	0.98	0.94	0.96	0.98
8		1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.89	0.94	0.94	0.05	0.97	0.93	0.94	0.96	0.98
9		1.00	1.00	1.00	0.99	1.00	1.00	0.96	0.96	0.98	0.94	0.94	1.00	0.97	0.98	0.94	0.96	0.98
10		1.00	1.00	1.00	0.92	1.00	0.93	0.88	0.93	0.98	0.84	0.86	1.00	0.97	0.97	0.89	0.92	0.96
11		1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.98	0.95	0.94	1.00	0.97	0.98	0.94	0.96	0.98
12		0.96	0.98	1.00	0.85	1.00	0.88	0.74	0.86	0.97	0.89	0.50	0.98	0.94	0.96	0.83	0.88	0.85
13		1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.98	0.95	0.94	1.00	0.97	0.98	0.94	0.96	0.98
14		1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.98	0.95	0.94	1.00	0.97	0.98	0.94	0.96	0.98
15		0.95	1.00	1.00	0.93	1.00	0.96	0.96	0.95	0.98	0.85	0.87	1.00	0.97	0.97	0.89	0.95	0.98
16		1.00	1.00	0.99	0.99	1.00	1.00	0.97	0.95	0.98	0.95	0.94	1.00	0.97	0.97	0.92	0.94	0.98
17		0.05	0.34	0.66	0.43	1.00	0.68	0.32	0.73	0.63	0.67	0.50	0.53	0.54	0.85	0.77	0.36	0.59
18		0.62	0.06	0.62	0.52	0.99	0.64	0.31	0.32	0.67	0.71	0.37	0.76	0.46	0.90	0.83	0.46	0.70
19		1.00	1.00	1.00	0.98	1.00	1.00	0.96	0.96	0.98	0.94	0.93	1.00	0.97	0.98	0.93	0.96	0.98
20		1.00	1.00	1.00	0.99	1.00	0.99	0.95	0.96	0.98	0.92	0.93	0.93	0.97	0.01	0.92	0.96	0.98
21		1.00	1.00	1.00	0.97	1.00	0.97	0.95	0.96	0.98	0.90	0.93	1.00	0.97	0.97	0.60	0.96	0.98
22		1.00	1.00	1.00	0.97	1.00	0.98	0.97	0.96	0.98	0.94	0.93	1.00	0.97	0.98	0.93	0.96	0.98
23		1.00	0.99	1.00	1.00	1.00	1.00	0.98	0.96	0.98	0.95	0.94	1.00	0.97	0.98	0.93	0.96	0.98
24		1.00	1.00	1.00	1.00	1.00	1.00	0.97	0.96	0.98	0.94	0.94	1.00	0.97	0.98	0.94	0.96	0.98
25		1.00	1.00	1.00	1.00	1.00	1.00	0.97	0.96	0.98	0.95	0.94	1.00	0.97	0.98	0.94	0.95	0.98
26		1.00	1.00	1.00	1.00	1.00	1.00	0.97	0.96	0.98	0.95	0.94	1.00	0.97	0.98	0.90	0.96	0.98
27		1.00	1.00	1.00	1.00	1.00	1.00	0.97	0.96	0.98	0.95	0.94	1.00	0.97	0.98	0.94	0.96	0.98
28		1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.96	0.98	0.94	0.94	1.00	0.97	0.98	0.93	0.96	0.98
29		1.00	1.00	1.00	1.00	1.00	1.00	0.97	0.96	0.98	0.93	0.94	1.00	0.97	0.98	0.94	0.96	0.98
30		1.00	1.00	1.00	0.98	1.00	0.98	0.95	0.95	0.98	0.91	0.94	1.00	0.97	0.98	0.93	0.95	0.98
31		1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.98	0.95	0.94	1.00	0.97	0.98	0.94	0.96	0.98
32		1.00	1.00	1.00	0.99	1.00	0.99	0.96	0.96	0.98	0.94	0.94	1.00	0.97	0.98	0.83	0.96	0.98
33		1.00	0.98	1.00	1.00	1.00	1.00	0.97	0.96	0.98	0.95	0.94	1.00	0.97	0.98	0.94	0.96	0.98
34		1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.98	0.95	0.94	1.00	0.97	0.98	0.94	0.96	0.98
35		1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.98	0.95	0.94	1.00	0.97	0.98	0.94	0.96	0.98
36		1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.98	0.95	0.94	1.00	0.97	0.98	0.94	0.96	0.98
37		1.00	1.00	1.00	0.98	1.00	0.99	0.95	0.96	0.98	0.94	0.92	1.00	0.97	0.97	0.91	0.95	0.97
38		1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.98	0.95	0.94	1.00	0.97	0.98	0.94	0.96	0.98
39		1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.98	0.95	0.94	1.00	0.97	0.98	0.94	0.96	0.98
40		1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.98	0.95	0.94	1.00	0.97	0.98	0.94	0.96	0.98
41		1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.98	0.95	0.94	1.00	0.97	0.98	0.93	0.96	0.98
42		1.00	1.00	1.00	1.00	1.00	1.00	0.97	0.96	0.98	0.96	0.94	1.00	0.97	0.98	0.92	0.96	0.98
43		0.01	1.00	1.00	0.96	0.94	0.95	0.93	0.95	0.98	0.93	0.93	1.00	0.97	0.98	0.92	0.96	0.98
44		1.00	1.00	1.00	1.00	1.00	0.00	0.97	0.96	0.98	0.94	0.95	1.00	0.97	0.98	0.92	0.96	0.98
45		1.00	1.00	1.00	0.98	1.00	0.95	0.94	0.95	0.98	0.89	0.90	1.00	0.97	0.98	0.91	0.96	0.96
46		1.00	1.00	1.00	1.00	1.00	1.00	0.97	0.96	0.98	0.95	0.94	1.00	0.97	0.98	0.92	0.96	0.98
47		1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.98	0.95	0.94	1.00	0.97	0.98	0.94	0.96	0.98
48		1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.96	0.98	0.95	0.94	1.00	0.97	0.98	0.94	0.96	0.98
49		0.97	0.06	0.56	0.58	1.00	0.75	0.38	0.35	0.56	0.51	0.37	0.69	0.96	0.83	0.81	0.41	0.55
50		1.00	1.00	0.00	0.97	1.00	0.99	0.94	0.95	0.86	0.90	0.94	0.98	0.97	0.88	0.95	0.96	0.98
51		1.00	1.00	0.99	0.01	1.00	1.00	0.97	0.95	0.98	0.94	0.94	1.00	0.97	0.97	0.92	0.94	0.98

Figure 5: Probability of detection in correspondence to persistent failures of a single sensor. Cells are colored based on the probability of detection as follows: dark red for probability less than 20%; light red for probability between 20% and 40%; yellow for probability between 40% and 60%; light green for probability between 60% and 80%; dark green for probability greater than 80%.

6.2 Fault Detection, Diagnosis and Prediction

6.2.1 Expanding the Open-Source Library popt4jlib with Parallel Deep NN Training Capabilities

The QARMA algorithm relies on a set of foundational classes in the "parallel" and "parallel.distributed" packages of the Open-Source project popt4jlib (https://github.com/ioannischristou/popt4jlib) for its parallel/distributed processing needs. Given the strong potential of Deep Learning techniques for highly accurate predictions in use-cases for which FIREMAN applies (see below sections), within the popt4jlib library, we have developed a package called "neural" that implements most standard –as well as non-standard– algorithms for feed-forward neural networks of arbitrary depth. In Figure 6, we show the structure of this package as a standard UML class diagram.

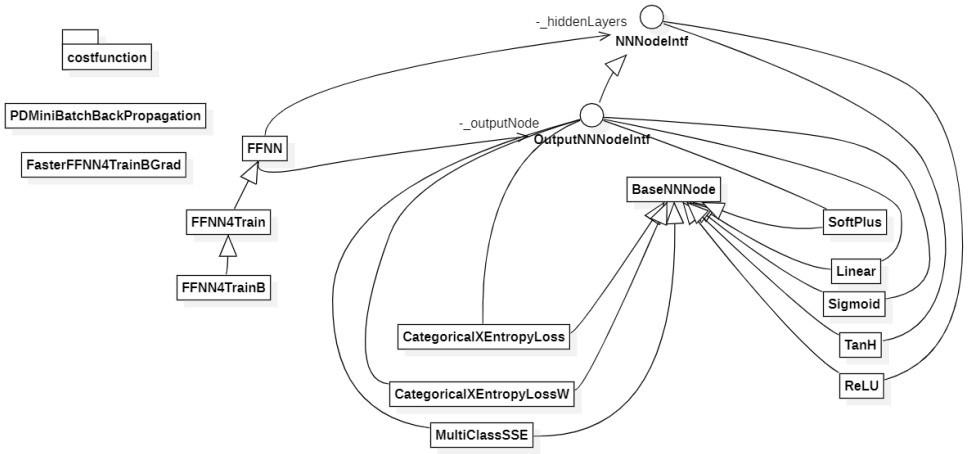

Figure 6: Static UML Class Diagram of "neural" Package in popt4jlib

The package implements a number of advanced features for training ANNs of arbitrary depth:

- during training, utilizes any number of cores available via a user-defined input parameter: the selected batch is broken up into as many pieces as there are threads, and each thread computes the gradient of the piece assigned to it. The overall gradient is then computed as the sum of the gradients of each piece.
- allows for various strategies for the selection of the batches in each major iteration.
- allows for network architectures where different nodes with different activation functions can mix in the same layer.
- uses a Spring-style object creation factory that allows the neural network topology to be described by the user in a properties file that allows for easy experimentation without any code modification or recompiling.

- allows for substituting the standard Stochastic Gradient Descent algorithm for optimizing the objective cost function with any of the optimization algorithms for unconstrained or box-constrained Non-linear Programming (NLP) developed in the library, thus allowing for easy experimentation with other gradient-descent based algorithms such as Conjugate Gradients, Armijo-rule algorithms for gradient descent with line-search, meta-heuristics and so on.

Experiments with this package are detailed in the computational results section.

6.2.2 Autoencoders for Fault detection

We will extend the general framework from Sec. 6.1 to application of autoencoders for fault detection and interpretation. Our focus will be placed on (i) the interpretation of faults, (ii) re-usability of the approach in different CPS scenarios, and (iii) performance assessment. To this end, we aim to build upon the works of [7] and [8], which introduce the sparse identification of nonlinear dynamics and the Long Short Term Memory (LSTM) autoencoder for fault detection, respectively.

7 Computational Results

7.1 Comparing the Parallel MERSFinder Algorithm with SCIP v.7

As mentioned in Sec. 4.2, we have implemented a parallel hybrid Breadth-First Search / Beam Search procedure to compute a "small" subset of rules that cover the majority of instances in a training dataset covered by the totality of rules extracted by QARMA applied on it. In table 1, we present the running times on a modern server equipped with a 12-core intel i9 core hyper-threaded CPU with 64GB RAM, and compare them with the run-time of the state-of-the-art Open Source MIP solver SCIP version 7. The test-bed against which we compare the two algorithms is based on the synthetic grid-fault dataset described in [9]. QARMA produced $23,810$ non-dominated rules, covering 100% of the training data. For values $a = 0.9$ and $S = 33$, SCIP solved the Integer Programming problem (MERS) in $19,281$ seconds which, even though this is an offline process, it is still a long time to solve what is essentially a feasibility problem.

Table 1: Comparing MERSFinder with SCIP v.7

#Threads	#Rules in solution	Response time (secs)	Ratio MERSFinder/SCIP time
1	32	7861	2,45
2	32	2877	6,70
4	32	1463	13,18
8	32	729	26,45
12	33	364	52,97
16	32	369	52,25
24	30	80	241,01

It is easy to see that our parallel hybrid search algorithm can outperform the currently fastest Open Source MIP solver by more than two orders of magnitude $(241\times)$ when 24 threads are used to speed up the search.

7.2 Experiments with QARMA and Deep Neural Networks on Synthetic Power Grid Fault Diagnosis Datasets

Next, we turn our attention to a comparison of the performance of QARMA versus Deep Neural Networks (as implemented in the popt4jlib library). As base data we use the datasets described in [9].

In this experiment, first, we simulate the system and a apply a DM-DFT (Delta-Method Discrete Fourier Transform) and then this method is used find the fault samples where the features (currents and voltages in the dataset) will be extracted. The DFT maps a given point of the input signal (i.e., current or voltage) into two points in the output signal. For N samples, considering the pair x_n (input signal) and X_k (its DFT)

$$X_k = \sum_{n=0}^{T-1} x_n e^{-2\pi i k n/T}, \tag{1}$$

where $0 \leq k \leq T-1$, and T is the number of samples per cycle and n represents current phase (a,b or c). The DM-DFT uses a moving window of length T instead of the complete signal, thus allowing faster fault recognition. Sample rate is 4 kHz, about 80 samples per cycle (which is usually used in commercial relays). To obtain a highly accurate signal point 1.5 cycles or about 120 samples after fault occurrence are needed. When a transmission line is in faulty state, magnitudes of currents and voltages (which are the features used in the fault classification task) can suddenly change depending on the type of fault and its characteristics. The generated dataset was split into two subsets: 75% of a random shuffle of the dataset was kept for training and the remaining 25% was used to validate the accuracy of the trained models. The exact same split was used for all simulations with all different algorithms. Table 2 shows the results of running several well-known machine learning algorithms for supervised learning on the produced dataset; the total set of classifiers shown were initially chosen based on their popularity and/or prior effective use in the field of fault diagnosis, and Fig. 7 shows the results for the classification task in terms of accuracy of fault detection alone. In this particular dataset, accuracy of classification is a reasonable metric, because the dataset comprises of many classes (10 faults plus the "no-fault" class) but is otherwise almost perfectly balanced. Because of this, more advanced metrics such as Area-Under-the-Curve (AUC) or F1-score are not shown here.

With the exception of Naive Bayes and (more surprisingly) of AdaBoost, the other classifiers we compared against, produce rather decent classification results; excluding Logistic Regression, all the other classifiers obtained an accuracy higher than 85% on the test set. However, our deep neural network and the QARMA-produced rule ensemble stand out with classification performance at 98% or more, justifying our choices.

The accuracy achieved with the deep learning model setup was remarkably high, 98.33%. It was achieved by a 4-layer (4L) deep network, with 10 nodes in the first hidden layer (5 linear and 5 SoftPlus units), 5 SoftPlus units in the second layer, and 5 SoftPlus units in the third layer; the output layer had 11 sigmoid units corresponding to each of the 11 fault class types. The total cost function of the network was the sum of square errors of each output node over all training instances. The entire network was trained via stochastic gradient descent (SGD) as the weights optimization algorithm. The backpropagation algorithm was used to compute the overall function gradient. The open-source library popt4jlib (https://github.com/ioannischristou/popt4jlib) was used to train this network.

The SGD method was used with a mini-batch size of 50 instances. In addition, normalization of the gradient vector $g(w) = \nabla E(w)$ to 1 before the steepest gradient descent rule $w \leftarrow w - \alpha g(w)$ was important for quick convergence; the learning rate α decayed as the epochs progressed according to the formula $\alpha \leftarrow \alpha 500/(500+epoch)$. The remarkable validation accuracy was achieved after only 10 epochs in less than 8.6 seconds of wall-clock training time on an Intel i-9 10920X processor, using all its 24 logical cores. This high accuracy is due to the large size of the simulated fault dataset and equally importantly because of the *balance between the sample sizes of the various classes*. The strong success of the DL model is also because all the voltages and currents from both lines were available, including neutral currents. The faults that were not selected properly were all single-phase-to-ground faults. This can be explained as follows: in those fault cases where the fault resistance took the larger value, only one of the phases changed slightly compared to the other two phases making the feature variation difficult for the model to detect.

Table 2: ML results on the fault-grid dataset.

Classifier	Dec. Tree	1-L ANN	4-L ANN	SVM	Ripper-k	Naïve Bayes	Log Regr.	AdaBoost.M1	QARMA
Accuracy	94.62%	95.18%	**98.33%**	89.05%	86.17%	59.42%	78.47%	17.81%	**98%**

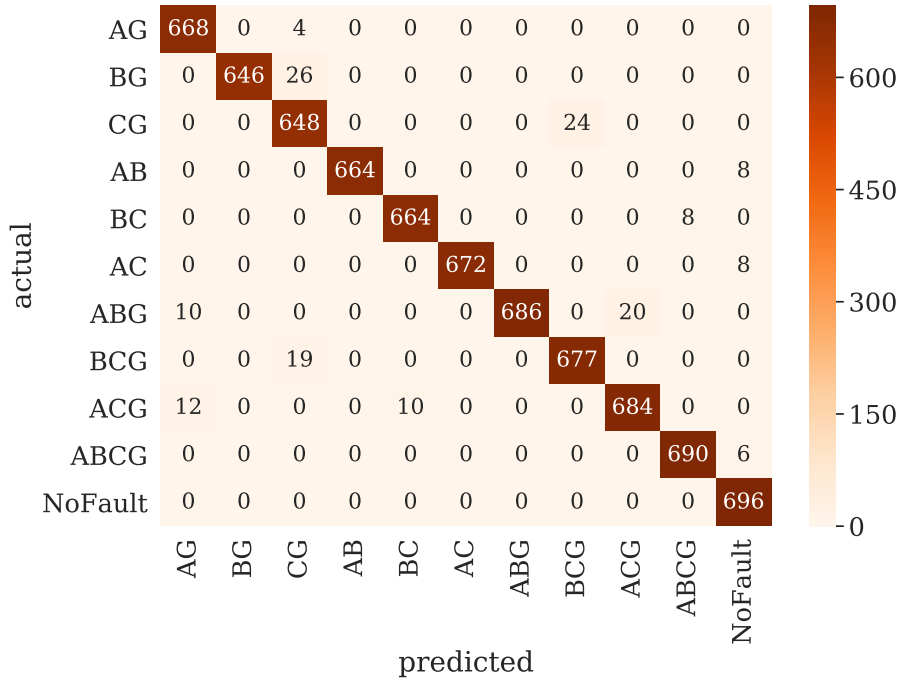

Figure 7: Fault classification via deep neural network confusion matrix.

Table 3: Feature selection on original dataset.

Round	1	2	3	4	5	6	7	8
Feature	$L_{I_A}, L_{I_B}, L_{I_C}, L_{I_R}$	$R_{I_A}, R_{I_B}, R_{I_C}, R_{I_R}$	$L_{I_A}, L_{I_B}, L_{I_C}, L_{V_A}, L_{V_B}, L_{V_C}$	$R_{I_A}, R_{I_B}, R_{I_C}, R_{V_A}, R_{V_B}, R_{V_C}$	$L_{I_A}, L_{I_B}, L_{I_C}$	$R_{I_A}, R_{I_B}, R_{I_C}$	$L_{V_A}, L_{V_B}, L_{V_C}$	$R_{V_A}, R_{V_B}, R_{V_C}$

With this setup, the importance of availability of all features was tested. Table 3 lists the number of features tested, and Fig. 8 shows the results with an ANN.

With fewer features, the ANN does not perform as well, emphasizing the importance of neutral current estimation. However, when only one-end currents are available, the validation error of the algorithm is still adequate for the task.

We ran QARMA on the same training set with the user-defined support threshold of 3.5% and the confidence threshold of 90% to obtain 5333 rules covering 97.8% of the entire training set. Then, a slight variant of the decision making algorithm described in the previous section, based on weighted voting, was used: for each instance in our test set, as long as the instance is covered by more than 100 rules, the instance's class is decided upon by the majority vote of the *top 10 firing rules having the highest confidence on the training set; instances that fail the minimum coverage requirement are not classified*. This algorithm resulted in high

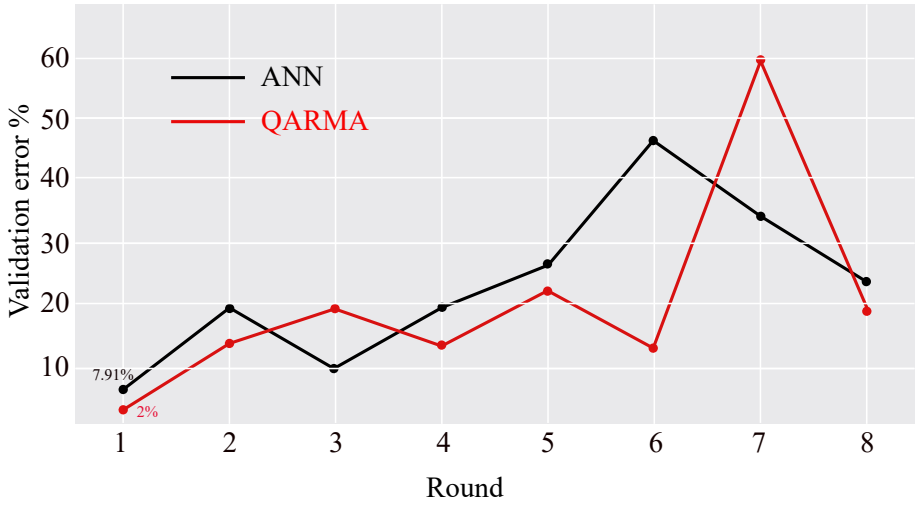

Figure 8: Validation error obtained with same model parameters using less features.

accuracy comparable with the one obtained by the DL, around 98% but at the following cost: a longer training time (around 15 minutes of wall clock time on the same intel core i9 10920X CPU with 24 logical cores.) For a small percentage of testing instances, approximately 4%, QARMA was not able to provide a decision, because of the small number of rules firing on them. However, we expect that QARMA and its decision-making components will compare equally well or even outperform deep learning techniques in training sets that are more highly skewed.

Another advantage of QARMA relates to the sensitivity of the produced rules with respect to noise in the data. We already saw that when the training and testing data suffer from Gaussian white noise with $\sigma = 100$ the performance of the NN drops just below 80%. On the other hand, when QARMA ran on the same noise-polluted training dataset with $\sigma = 100$, and then the resulting rule-ensemble asked to classify an equally noise-polluted testing dataset (with $\sigma = 100$), surprisingly, QARMA performance remained very high at 97.1% making QARMA much more robust to noise in the measurements than the NN.

Even though more research and experiments are needed to fully explain why this might happen, we believe that the answer about the cause of the difference of robustness between the two classifiers is probably lying on the underlying models' complexity: the NN being a deeply composite function of many variables (connection weights and bias thresholds) when optimized on a noise-polluted training dataset is easier to over-fit, and "learn" some of the noise in its weights. On the other hand, QARMA being a rule extractor that learns rules that have only a small number of different features in their antecedent conditions, provides an ensemble of simple if-then decision rules that are more likely to hold true in the presence of noise.

On the other hand, QARMA produces a model with a set of quantitative rules that are much easier to understand and reason about than most of the other models, and DL models in particular; this makes QARMA results much easier to explain to humans than any other model. Every extracted rule is trivially checked against the training dataset for validation purposes, and it is also trivial to understand "what it means" since the preconditions of the rule are nothing more than a conjunction of the restrictions of the attributes that comprise the rule's antecedents to certain intervals. This ease of understanding of rules is what has

made them particularly attractive since the beginning of AI and ML research. In fact, already since the 1980s, there have been attempts to extract the knowledge that is embedded in neural network models into sets of rules [10] since such rule sets were recognized from the beginning as the most obvious knowledge representation that can exist. Therefore, QARMA is, in general, a particularly good fit for the newly emerging "eXplainable Artificial Intelligence" (XAI) paradigm, the term "explainable" meaning that the resulting model that the algorithm produces can be easily understood by humans.

7.3 Batch, Stream-based and PyTorch Deep Learning Experiments

Experiments related to Sections 5, 6.1, and 6.2.2 will be performed on Juno server at the CTTC, CSC (Finish Super-Computing Facility), and on Google CoLab. Results and scripts with TEP and other synthetic datasets will be made publicly available on the github and in the form of published papers.

8 Conclusions and future directions

We have described in detail the existing algorithmic infrastructure on which we are currently building the FIREMAN detection and prediction techniques for rare events in Industry 4.0 settings. We have analyzed our two main approaches, based on rule-extraction systems and deep learning systems from large datasets, and we have described in detail the progress made so far into the project. We presented computational results from a number of diverse datasets that show that both approaches hold significant promise towards our goal.

Depending on the conducted data analysis and performance assessment, future research directions will involve -among others- the:

- Addition of secondary features for improvement of prediction results (rolling window average/variance/entropy etc.) and subsequent assessment;
- Definition and computation of uncertainty in the implemented deep learning topologies to measure level of general applicability of the designed networks.

In the FIREMAN project proposal, we have outlined a research approach where we shall use the basic ideas of the so-called Template-Grid method to quickly identify repeating patterns in the shape of a time-series that exhibit high correlation with certain events we wish to forecast, for example a machine or component failure. Statistical analysis work in other projects (e.g. PROPHESY, https://www.prophesy.eu) has shown that indeed, even a single sensor variable can often become a good indicator of a problematic component hours before human operators can realize that there is something wrong in a production run. We therefore expect that techniques based on the Template-Grid method will be able to quickly determine such patterns even in very large datasets, since they work with single variables (features) at a time. We shall begin work on this approach within 2021.

References

[1] I. T. Christou, E. Amolochitis, and Z.-H. Tan, "A parallel/distributed algorithmic framework for mining all quantitative association rules," *arXiv preprint arXiv:1804.06764*, 2018.

[2] I. T. Christou, "Avoiding the hay for the needle in the stack: Online rule pruning in rare events detection," in *16th International Symposium on Wireless Communication Systems (ISWCS)*. IEEE, 2019, pp. 661–665.

[3] I. T. Christou, N. Kefalakis, A. Zalonis, and J. Soldatos, "Predictive and explainable machine learning for industrial internet of things applications," in *Proceedings of the 16th IEEE Intl. Conf. On Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2020, pp. 213–218.

[4] I. T. Christou, N. Kefalakis, A. Zalonis, J. Soldatos, and R. Bruchler, "End-to-end industrial iot platform for actionable predictive maintenance," in *Proceedings of the 4th IFAC Workshop on Advanced Maintenance Engineering Services and Technologies*, 2020.

[5] J. T. McCoy, S. Kroon, and L. Auret, "Variational autoencoders for missing data imputation with application to a simulated milling circuit," *IFAC-PapersOnLine*, vol. 51, no. 21, pp. 141–146, 2018.

[6] J. Yoon, J. Jordon, and M. Van Der Schaar, "Gain: Missing data imputation using generative adversarial nets," *arXiv preprint arXiv:1806.02920*, 2018.

[7] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton, "Data-driven discovery of coordinates and governing equations," *Proceedings of the National Academy of Sciences*, vol. 116, no. 45, pp. 22 445–22 451, 2019.

[8] P. Park, P. D. Marco, H. Shin, and J. Bang, "Fault detection and diagnosis using combined autoencoder and long short-term memory network," *Sensors*, vol. 19, no. 21, p. 4612, 2019.

[9] D. Goutierrez-Rojas, I. T. Christou, D. Dantas, A. Narayan, Y. Y., and P. Nardelli, "Performance evaluation of machine learning for fault selection in power transmission lines," *Under Peer Review*, 2020.

[10] G. G. Towell and J. Shavlik, "Knowledge-based artificial neural networks," *Artificial Intelligence*, vol. 70, no. 1–2, pp. 119–165, 1994.