Available online at www.sciencedirect.com
ScienceDirect
Computers & Electrical Engineering 00 (2020) 1–13
COMPELECENG

Imputation of Missing Values Affecting the Software Performance of Component-based Robots

Nuño Basurto[0000−0001−7289−4689], Ángel Arroyo[0000−0002−1614−9075], Carlos Cambra[0000−0001−5567−9194], Álvaro Herrero[0000−0002−2444−5384]

Grupo de Inteligencia Computacional Aplicada (GICAP), Departamento de Ingeniería Informática, Escuela Politécnica Superior, Universidad de Burgos, Av. Cantabria s/n, 09006, Burgos, Spain.

{*nbasurto, aarroyop, ccbaseca, ahcosio*}*@ubu.es*

Abstract

Intelligent robots are foreseen as a technology that would be soon present in most public and private environments. In order to increase the trust of humans, robotic systems must be reliable while both response and down times are minimized. In keeping with this idea, present paper proposes the application of machine learning (regression models more precisely) to preprocess data in order to improve the detection of failures. Such failures deeply affect the performance of the software components embedded in human-interacting robots. To address one of the most common problems of real-life datasets (missing values), some traditional (such as linear regression) as well as innovative (decision tree and neural network) models are applied. The aim is to impute missing values with minimum error in order to improve the quality of data and consequently maximize the failure-detection rate. Experiments are run on a public and up-to-date dataset and the obtained results support the viability of the proposed models.

Keywords: software component, intelligent robots, anomaly detection, missing values, supervised learning, regression

1. Introduction

Wide attention has been devoted to the development of intelligent robots in recent years. Although significant contributions have been done, it still is a challenging field where further progress is required to satisfy present and future demands. One of such demands is the fluent interaction with non-expert humans, that is required for robotic systems to be widely integrated in a variety of homes and workplaces [1]. In order to get such fluency, performance of both hardware and software is a keystone. However, the ever-increasing complexity of robots leads to a parallel increase in chances of experiencing a failure. Accurate and prompt detection of such events is required in order to improve performance and hence fluency. Full attention has been payed to advance in many subfields of the robotics arena but according to some authors [2], further effort must be devoted to anomaly detection in such systems. It is even more challenging when failures happen in a real-world context where complex phenomenon may interfere.

Accordingly, present paper focuses on the preprocessing of robot-performance data, whose importance is widely acknowledged. More precisely, the aim is the successful imputation of Missing Values (MV) in order to get as much data as possible for subsequent anomaly/failure detection. Thus, a wide variety of Artificial Intelligence (AI) models are applied in order to predict the MV of all the dataset components.

The successful detection of anomalies/faults is a challenging task that does not only apply to robots [3] [4] [5]. From a business perspective [6], AI in general, and Machine Learning (ML) in particular, can greatly contribute to anomaly detection and some other interesting tasks, maximizing companies benefits.

Since pioneer works [7] in the application of ML to robotics, unsupervised [8], supervised, and reinforcement [9] learning models have been previously applied. A variety of problems have been addressed so far such as control [10] [11] and communications [12] among others. In the case of anomaly detection, most ML previous work has been focused on the detection of hardware anomalies [13], while software anomalies have been scarcely investigated. Software failures often occur in robotic systems and their automatic detection requires training data. The problem comes from the difficulty of obtaining the data either because of the lack of execution traces or because the existing registers do not refer to the exact moment in which anomalies are produced. That is why it is difficult to find a dataset generated in a controlled environment where all the information is available. Furthermore, when data are gathered in a real-life environment, quite likely there will be some or many MV, that can not be processed by ML models.

One of the few works on the detection of software anomalies within the framework of component-based robots is [14]. In that paper, authors proposed the only publicly-available dataset (further details in section 3) that gathers data from different performance indicators of a robot. The dataset [15] has been used in present paper as a benchmark dataset due to its interest and novelty. In this dataset, there are many MV associated to different data so a robust strategy must be followed in order to deal with them as most ML models can not process such data. One of the obvious preprocessing alternatives in order to solve such problem in the data is removing MV, either by deleting data instances or by deleting attributes. However, a more advanced proposal is to impute such values, keeping some information that could be useful for the subsequent anomaly detection. This approach is adopted in the present paper.

AI methods have been previously applied for imputation of MV [16]. However, scant attention has been devoted to the application of ML methods in order to solve such problem in robot datasets. One of the very few previous proposals is [17], where a probabilistic approach for classification using incomplete data was applied. The author performed a classification (for failure detection) of data samples by calculating the a priori probability of MV, determined from the data samples that are not missing. However, the author proposal was only applied to outdated (1999) datasets containing hardware anomalies. Differentiating from previous work, the present paper is the first approach to impute MV in a dataset containing information about the performance of the software components of a robot. A comprehensive benchmark comprising a wide variety of methods has been performed and some of the methods are applied to this problem for the first time.

The methods applied for imputation of MV are introduced in section 2, while the analysed case study is described in section 3. The performed experiments, together with their associated results are compiled in section 4. Finally, the main conclusions and some proposals for further work are presented in section 5.

2. Imputation Methods

As previously stated, ML methods are applied in present study for imputation of MV. More precisely, experiments have been run with four regression techniques and two Artificial Neural Network (ANN) models with different training algorithms. The applied techniques are described in the following subsections.

2.1. Regression Techniques

Regression tries to model the relationship between two variables in the dataset by fitting a linear equation to the input data. One of the variables is the predictor variable and the other one is considered to be the criterion variable [18]. The general purpose of multiple regressions is to learn more about the relationship between several independent or predictor variables and a dependent or criterion variable. Such relationships can be linear or non-linear, leading to the two techniques that are described below.

2.1.1. Linear Regression

Linear Regression (LR) attempts to model the relationship between two or more explanatory variables and a response variable by fitting a linear equation to the dataset [19]. Every value of the predictor variable (x) is associated with a value of the criterion variable (y). The regression line for p explanatory variables (x_1, x_2, ..., x_n) is defined as follows:

$$U_y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n \quad (1)$$

This line describes how the mean response U_y changes with the explanatory variables. The observed values for Y vary about their means U_y and are assumed to have the same standard deviation σ. The fitted values b_0, b_1, ..., b_n estimate the parameters $\beta_0, \beta_1, ..., \beta_p$ of the population regression line. Since the observed values for y vary about their means U_y, the multiple regression models include a term for this variation. The model is expressed as DATA = FIT + RESIDUAL, where the "FIT" term represents the expression $\beta_0+\beta_1x_1+\beta_2x_2+ ... +\beta_nx_n$. The "RESIDUAL" term represents the deviations of the observed values (y) from their means U_y, which are normally distributed with mean 0 and variance σ. The notation for the model deviations is ϵ. The model for linear regression, given n rows, is [19]:

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_n x_{in} + \epsilon_i \quad \text{for i=1, 2, ..., n} \tag{2}$$

2.1.2. Non-Linear Regression

Non-Linear Regression (N-LR) is a form of regression in which observational data are modeled by a function which is a non-linear combination of the input data and depends on one or more criterion variables . The parameters can take the form of an exponential, trigonometric, power, or any type of non-linear function. To determine the non-linear parameter values, an iterative algorithm is used. The model can be defined as:

$$y = f(X, \beta) + \varepsilon \tag{3}$$

Where B represents the non-linear parameter estimates to be computed, X is the dependent variables and ϵ represents the error terms.

2.1.3. Regression Trees

Regression Trees (RT) are usually shown growing upside down, with its root at the top. An observation passes down the tree through a series of splits (nodes). At them, a decision is made as to which direction (branch) to lead based on the value of one of the criterion variables. When a terminal node (leaf) is reached, a predicted response is given according to the end node. Trees are often built via binary recursive partitioning. In RT, values at the terminal nodes are assigned using the mean of cases in that node [20]. In present study, two variations of RT are applied:

- Fine Tree (FT). It usually comprises a reduced set of leaf nodes to optimize building time.
- The Boosting Ensemble (BE) algorithm iterative calls the regression-tree algorithm to construct an ensemble of trees. This ensemble combines results from many weak learners (least-squares boosting) with RT learners into one high-quality ensemble model. The response is calculated according to the Mean Squared Error (MSE) of the trained regression ensemble model that takes into account the results of boosting a high number (100) of trees.

2.2. Artificial Neural Networks

Artificial Neural Networks, also known as connectionist systems or adaptive networks, are simplified models of natural neural systems. The following definition, given by Hecht - Nielsen in 1988, formalizes the concept: *"An ANN is a parallel processing computer system distributed, consisting of a set of elementary processing units equipped with a small local memory and interconnected in a network through connections with associated weights. Each processing unit has one or more input connections and a single output connection that links to many collateral connections as desired. All processing associated with an elementary unit is a local, i.e. depends only on the values that take input signals from the unit and the internal state of the same"*. Two different models, adjusted to such definition, have been applied in the present study and are defined in the following subsections.

2.2.1. Multilayer Perceptron

The Multilayer Perceptron (MLP) consists of a system of simple interconnected neurons or nodes. They are connected by weights and output signals which are a function of the sum of the inputs to the node modified by a simple activation function. The architecture consists of several layers of neurons; the input layer serves to pass the input vector to the network. The terms called as "input vectors" and "output vectors" refer to the inputs and outputs of the MLP and can be represented as single vectors. The MLP may own al least one or more hidden layers and finally

an output layer. MLP are fully connected, with each node connected to every node in the next and previous layer. One of the critical issues of such model is the training (update) of all the weights as the error can be calculated in the output but weights in all layers must be updated. To solve such problem, backpropagation was proposed and several different algorithms implement it. Due to their known advantages, two of them have been applied in present study:

- Levenberg-Marquardt (LM) [21]. It is derived from the Newton's technique that is designed for minimizing functions that are sums of squares of non-linear functions.
- Bayesian Regularization (BR) [22]. It aims to improve the model's generalization capability, expanding the objective function with the addition of the sum of squares of the network weights.

2.2.2. Radial-Basis-Function Networks

In a Radial-Basis-Function Network (RBFN), each neuron in the hidden layer has its own centroid, and for each input vector $x = (x_1, x_2, \ldots, x_n)$, it computes the distance between x and the centroid. As a result, the output of these neurons is calculated as a non-linear function of this distance. Assuming that there are r input nodes and m output nodes, the overall response function without considering non-linearity in an output node has the following form:

$$\sum_{i=1}^{M} W_i * K\left(\frac{x - z_i}{\sigma_i}\right) = \sum_{i=1}^{M} W_i * g\left(\frac{\|x - z_i\|}{\sigma_i}\right) \tag{4}$$

where x is an input vector, $M \epsilon \mathbb{N}$ is the number of units in the hidden layer, $W_i \epsilon \mathbb{R}^m$ is the vector of weights linking the ith hidden-layer unit to the output nodes, K is a radially symmetric kernel function of a unit in the hidden layer, z_i and σ_i, are the centroid and smoothing factor of the kernel node, and g: $[0, \infty) \rightarrow \mathbb{R}$ is the activation function, which characterizes the kernel shape.

3. Real-life Case Study

Present research focuses on the imputation of MV in order to optimize anomaly detection in robot software. As previously stated, researchers at the University of Bielefeld (Germany) developed the only publicly-available [15] dataset [14] containing software anomalies. The analyzed robot has different components from different manufacturers integrated in the GuiaBot platform, developed by Mobile Robots. This robot was developed to participate in the RoboCup@Home competition. RoboCup@Home aims to deploy technology to address future robot-service in domestic contexts. The obtained score in such contest depends on two main issues [23]: the degree of autonomy and the performance of the robot. In the dataset under analysis, the robot faced several problems similar to those addressed by a human waiter, such as the identification of customers, serving drinks, and interacting with people and objects. The diverse nature of such tasks requires the robot to have different components to be attached to the main platform. Examples of such components are a mechanical arm, a sensor to detect people from their legs, and a camera to recognize people.

The different software components of the robot communicate through the Robotics Service Bus (RSB) middleware [24], using an event-based system and whose information is stored in a tool that incorporates the middleware called rsbag. The transferred information is encrypted as a notification. There is a framework called BonSAI whose responsibility is the combination between the sensors (that receive external information) and the actuators. For the representation and control of the execution flows, a Finite State Machine (FSM) is used. The general architecture of the robot is depicted in Figure 1, where it can be seen how the flow associated to the arm is different from that associated to the detection of legs because the second one does not lead to a physical action of the robot.

The two components analyzed in present research are the one for controlling the arm (ArmController) and that for the detection of legs (LegDetector). The first one performs two actions: to control the movement of the arm in different directions and to open and close the grip. The induced anomaly (ArmServerAlgo) increases the amount of movements to carry out the target tasks and hence penalizes the execution time. As a consequence, the robot performs a series of unnecessary actions that have a negative effect on the performance counters. On the other hand, the LegDetector component is responsible for recognizing and detecting the legs of human beings in order to avoid colliding with them. The associated anomaly (LegDetectorSkippable) causes the robot to perform the scan of legs a greater number

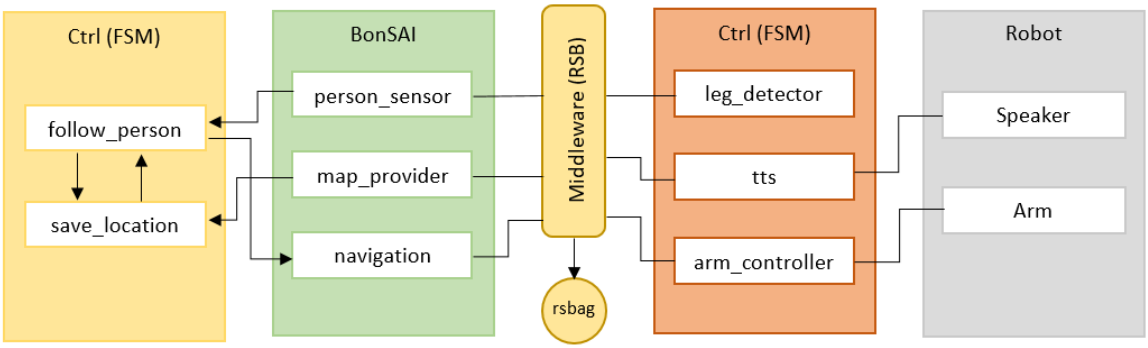

Figure 1. Robot system architecture comprising analyzed modules. Adapted from [14].

of times than needed. These components have been selected because, in the preliminary experiments carried out by the authors of the dataset [25], ArmController achieved the worst results while LegDetector achieved the best results when trying to detect the anomalies. Hence, the present study involves datasets that a priori have differences when performing classification tasks for failure detection. As it can be seen in Figure 1, these components are of different nature since one involves a physical response in the robot, while the other does not.

The whole dataset comprises data from several trials, being each one of them an attempt of the robot to perform a target task. The authors who gathered the data [14], detected that in some of these trials there were undetected anomalies, that's the reason why some of them are discarded. The analyzed data for each one of the components come from the performance counters, measured every second. Further details about the attributes comprised in the component datasets can be found in Table 1. This dataset is described in greater detail in [15].

Table 1. Explanation of the dataset attributes.

Variables	Description
vsize	The current size of virtual memory used by a task.
open_fds	The current number of file descriptors opened by a process.
rchar	Number of bytes that a process has read since the beginning.
open_connections	Number of network connections opened by a process.
stime	Amount of time of the process in user mode.
wchar	Number of bytes that a process has write since the beginning.
utime	Amount of time of the process in kernel mode.
num_threads	Number of threads that a process has in operation.
rss	Current RSS of a process
received_bytes	Number of bytes received by the interface.
sent_bytes	Number of bytes sent by the interface.
write_bytes	Number of bytes written by the device.

The induction of anomalies is not constant in the whole dataset, as anomalies are induced at a varying rate in the different trials. Table 2 shows the distribution of anomaly occurrences for the analyzed components. There are 10 trials associated to the ArmController and 12 to the LegDetector components. All of the trials have been fused in present study to generate 2 different datasets, one per component.

There is a similar rate of MV in the two component datasets. In the ArmController one there are 26025 (8.5%) data instances containing missing values while there are 24163 (10.03%) in the case of the LegDetector.

Table 2. Occurrences of each anomaly and distribution per trials.

Anomaly	1 time	2 times	3 times
ArmServerAlgo	28, 32, 36, 41, 45, 57, 65, 71	21	23
LegDetectorSkippable	19, 21, 24, 31, 36, 55, 57, 64, 66, 68, 70		71

4. Experiments and Results

The regression techniques described in Section 2 have been applied to the datasets (ArmController and LegDetector) detailed in Section 3, in order to evaluate their imputation capability on all the attributes of each dataset. To get more significant results, they are validated by the well-known n-fold Cross-Validation (CV) scheme. CV is a technique that splits the data, in order to measure the performance (MSE in the present study) of each technique in different subsets of data. The number (n) of data partitions has been set to 10 for all the experiments in the present study, as it is a standard value.

In order to do the regression on all the attributes of the 2 datasets, different variations have been generated for each dataset, one per each attribute. As a result, 11 variant datasets have been generated for the ArmController dataset and 8 for the LegDetector. In each case, the attribute on which the regression is applied is stated as the target column while the remaining ones are the predictor variables.

In the case of the ANN models, the training process has been repeated 10 times for each training algorithm (LM and BR for the MLP). The main purpose of this repetition is to reduce the effect of randomness and therefore obtaining more representative results. A sigmoid activation function in the hidden layer and a linear activation function in the output layer have been applied in the case of the MLP. A radial-basis transfer function was applied in the case of RBFN. Additionally, different parameters have been tested for some of the techniques. However, for the sake of brevity, only the results obtained for the following parameters are shown in Sections 4.1 and 4.2:

- FT: minimum leaf size (4).
- BE: minimum leaf size (8), and number of learners (30).
- RBFN: spread (40) and maximum number of neurons (10).
- MLP (both LM and BR): neurons in the hidden layer (10) and training epochs (70).

The average MSE and execution time (for the 10 folds) have been calculated for all the experiments and are shown in Tables 3 to 7. In the case of the MLP, the mean and standard deviation for the 10 executions are shown. MSE is calculated when trying to impute 25% of the data samples (considered as the MV) for each one of the CV folds, while 75% of the data samples are used to build/train the methods.

4.1. ArmController Component

Tables 3 and 4 show the results obtained for the ArmController component. More precisely, Table 3 shows the average MSE obtained by each one of the regression techniques when predicting values for each one of the attributes in the original dataset. After analyzing results in this table, it is worth mentioning that figures significantly vary depending on the applied method and the target attribute. The N-LR method gets the best results (minimum error) for 9 out of the 11 components, while 3 other methods get the best result only in one case; FT for the open_fds component, RBFN for the num_threads component, and MLP with the BR training algorithm for the vsize component.

The differences in the MSE values are quite big, being the FT (except for the open_fds component) and BE methods the ones with the worst performance in terms of MSE.

When considering the attribute which obtain a lower MSE, the open_fds achieves the best results while the open_connectins is the second one. On the other hand, the vsize is the attribute that obtain the higher error for all the methods.

Table 3. Average MSE value per method and dataset attribute on the ArmController component.

	LR	N-LR	FT	BE	RBFN	MLP_LM		MLP_BR	
						Mean	Std Dev	Mean	Std Dev
vsize	1.67E-06	1.64E-06	1.42E-05	3.12E-04	1.67E-06	3.49E-09	9.25E-26	2.89E-09	0
open_fds	2.80E-23	2.76E-23	3.52E-24	2.43E-12	2.79E-23	7.97E-23	1.97E-39	1.28E-22	5.26E-39
rchar	5.35E-19	5.28E-19	4.83E-10	5.67E-10	5.35E-19	1.50E-15	0	8.54E-16	2.20E-32
open_connections	2.80E-23	1.74E-28	1.31E-14	2.21E-12	2.79E-23	2.31E-22	2.63E-39	2.43E-22	0
stime	1.26E-17	1.26E-17	2.65E-09	2.78E-09	1.26E-17	3.32E-14	0	3.89E-14	0
wchar	6.63E-19	6.52E-19	3.55E-10	4.19E-10	6.63E-19	8.67E-16	0	1.15E-15	4.69E-32
utime	2.17E-14	2.16E-14	4.28E-08	4.97E-08	2.17E-14	6.31E-12	2.82E-30	2.01E-11	2.26E-29
num_threads	5.59E-22	1.43E-22	5.03E-23	1.23E-11	5.59E-22	2.99E-21	7.36E-38	5.66E-21	0
rss	7.44E-11	2.48E-11	1.93E-06	5.82E-06	7.44E-11	6.73E-10	8.67E-27	9.39E-10	0
received_bytes	1.67E-17	1.57E-17	9.95E-10	2.10E-09	1.67E-17	5.19E-14	5.29E-31	1.04E-13	8.82E-32
sent_bytes	5.13E-18	4.83E-18	6.38E-10	1.22E-09	5.13E-18	3.02E-14	1.76E-31	1.61E-14	7.61E-31

Table 4. Average execution time per method and dataset attribute on the ArmController component.

	LR	N-LR	FT	BE	RBFN	MLP_LM		MLP_BR	
						Mean	Std Dev	Mean	Std Dev
vsize	8.13E-02	1.57E-01	1.44E-01	5.33E-01	2.00E-02	1.35E-01	1.20E-01	1.40E-01	1.17E-01
open_fds	3.08E-02	1.12E-01	2.08E-01	6.28E-01	3.34E-02	9.81E-02	2.57E-02	9.37E-02	1.30E-02
rchar	3.47E-02	1.75E-01	3.51E-01	1.03E+00	1.44E-02	9.68E-02	2.03E-02	9.60E-02	1.45E-02
open_connections	4.29E-02	1.74E-01	1.33E-01	5.48E-01	2.47E-02	1.04E-01	3.40E-02	9.90E-02	1.80E-02
stime	3.29E-02	1.45E-01	1.21E-01	5.57E-01	1.90E-02	9.76E-02	1.37E-02	1.06E-01	1.88E-02
wchar	3.33E-02	1.67E-01	1.66E-01	5.41E-01	3.88E-02	1.07E-01	2.20E-02	1.08E-01	2.42E-02
utime	3.65E-02	1.37E-01	1.34E-01	5.03E-01	1.32E-02	1.00E-01	1.23E-02	1.08E-01	1.91E-02
num_threads	3.43E-02	1.58E-01	9.89E-02	4.50E-01	1.45E-02	9.57E-02	1.23E-02	1.03E-01	1.52E-02
rss	3.41E-02	1.59E-01	1.87E-01	4.62E-01	1.29E-02	9.81E-02	1.63E-02	1.05E-01	1.17E-02
received_bytes	3.35E-02	1.39E-01	1.33E-01	5.36E-01	1.29E-02	9.78E-02	1.18E-02	1.11E-01	2.85E-02
sent_bytes	3.87E-02	1.49E-01	1.37E-01	5.83E-01	1.38E-02	1.06E-01	2.72E-02	9.61E-02	8.66E-03

For the MLP neural model (2 training algorithms), the standard deviation of the MSE is really low (zero in eight times). It means that obtain results are robust and consistent for the 2 training algorithms (LM and BR).

Complementary to the above-shown errors, Table 4 shows the average execution times for each method and attribute.

As far as execution times are concerned, it can be said that the results slightly vary. The RBFN neuronal model manages to be the fastest method for all the attributes. On the other hand, BE is the slowest method, as the highest execution times are obtained by this method. Analyzing the table by attributes, no big differences are appreciated in the time. However, the fastest results are obtained for the rss and received_bytes attributes.

In order to ease a visual analysis of these obtained results, Figure 2 and Figure 3 show the boxplots for the values that are summarized in Table 3 and Table 4 respectively. In Figure 2, all the results obtained by all the applied imputation methods are included for each one of the components. Similarly, in Figure 3, all the results obtained on all the dataset components are included for each one of the imputation methods.

It is worth highlighting from Figure 2 the great variability in the results when applying the different imputation methods to each one of the components. This is specially visible for the components open_fds (2), open_connections (4), and num_threads (8). For such components, the high error rates obtained by the BE method for the component open_fds and those obtained by the FT and BE methods for the other two components can be easily identified. The

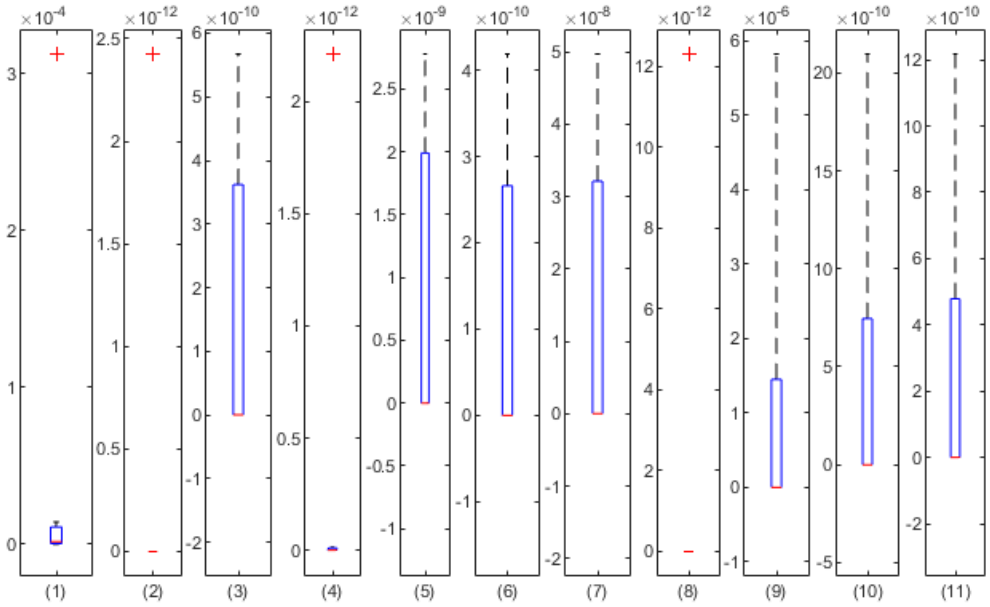

Figure 2. Boxplot of the MSE values (all imputation methods) on the ArmController component per attribute. (1) vsize, (2) open_fds, (3) rchar, (4) open_connections, (5) stime, (6) wchar, (7) utime, (8) num_threads, (9) rss, (10) received_bytes, (11) sent_bytes

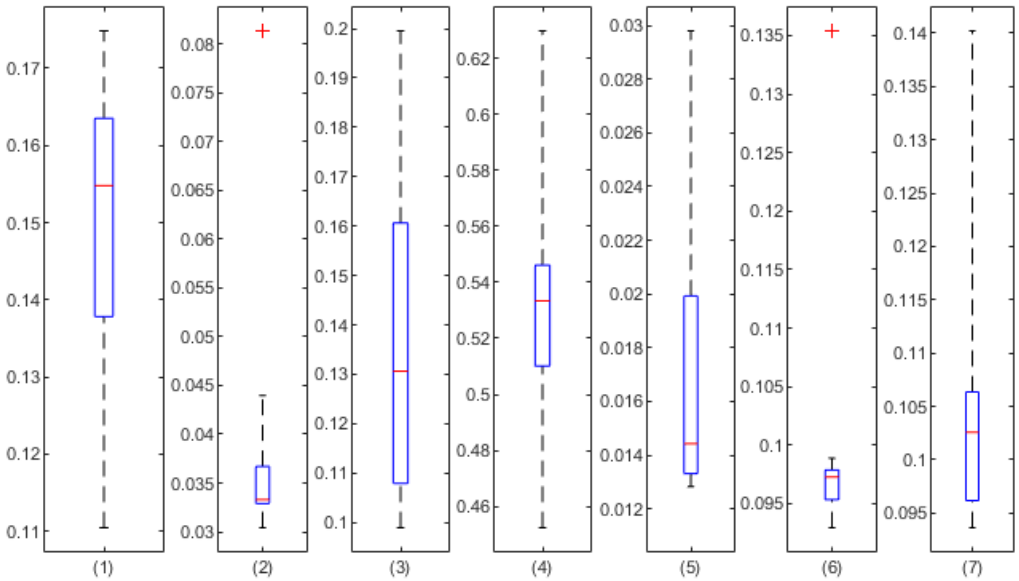

Figure 3. Boxplot of the execution time (all components) on the ArmController component per method. (1) LR, (2) N-LR, (3) FT, (4) BE, (5) RBFN, (6) MLP_LM, (7) MLP_BR

remaining error rates obtained for these components are shown in a small box, with no variance. Something similiar (to a certain extent) can be seen for the vsize (1) component. For the other components, error rates are pretty similar,

slightly varying between the different methods.

As previously mentioned, it can be seen in Figure 3 that there is a small variability in the execution times. In the case of the N-LR and MLP_LM methods, some outlier values can be observed, while the other values are very similar. All in all, average values are quite centered in the (25^{th} and 75^{th}) percentiles shown in the boxplot.

4.2. LegDetector Component

Similarly to what is shown above for the ArmController component (Subsection 4.1), MSE and execution times are shown for the experiments run on the LedgeDetector component. Therefore, Table 5 and Table 6 shows the MSE and execution times respectively obtained in the experiments on the LegDetector component.

Table 5. Average MSE value per method and dataset attribute on the LegDetector component.

	LR	N-LR	FT	BE	RBFN	MLP_LM		MLP_BR	
						Mean	Std Dev	Mean	Std Dev
write_bytes	1.07E-14	1.07E-14	8.42E-08	9.28E-08	1.07E-14	3.68E-11	0	2.8099E-11	4.06E-28
rchar	3.71E-12	3.71E-12	1.68E-06	1.73E-06	3.71E-12	6.87E-09	0	5.5068E-09	0
stime	1.83E-14	1.83E-14	1.02E-07	1.17E-07	1.82E-14	6.20E-11	9.03E-28	1.9076E-11	4.06E-28
wchar	9.66E-17	9.66E-17	1.70E-09	2.29E-09	8.74E-17	5.25E-14	1.76E-31	6.1249E-14	0
utime	1.11E-12	1.11E-12	4.12E-07	5.01E-07	1.08E-12	2.45E-09	4.62E-26	5.0123E-09	1.39E-25
rss	1.79E-05	1.79E-05	2.44E-03	3.48E-03	1.79E-05	1.31E-05	7.19E-08	1.2559E-05	4.68E-07
received_bytes	5.98E-14	5.98E-14	7.72E-08	1.25E-07	5.98E-14	5.83E-12	0	1.5439E-11	2.71E-28
sent_bytes	1.58E-14	1.57E-14	1.82E-08	2.71E-08	1.42E-14	8.29E-13	0	1.0276E-12	1.16E-29

Table 5 shows quite different results from those observed in Table 3 for the ArmController component (Subsection 4.1). In the case of the LegDetector component, the model that returns the best results (in terms of the MSE) is the RBFN neural model, for 7 out of 8 dataset attributes that have been imputed. For 3 of these attributes, namely write_bytes, rchar, and received_bytes, the LR and N-LR regression techniques have obtained similar error rates. The MLP with the BR training algorithm has obtained the best result for the rss component, while the FT and BE trees have performed poorly (similarly to what is shown in Table3). As for the previous component, the standard deviation for the two MLP training algorithms (10 executions each) are really low; it amounts to zero in 6 out of the 16 cases that are shown. Analyzing the MSE per components (Table 5), wchar is the one for which the lowest error rate has been obtained. Secondly, similar MSE values have been obtained for the write_bytes, stime, received_bytes and sent_bytes attributes. Oppositely, rss is by far the one with the highest MSE value.

Table 6. Average execution time per method and dataset attribute on the LegDetector component.

	LR	N-LR	FT	BE	RBFN	MLP_LM		MLP_BR	
						Mean	Std Dev	Mean	Std Dev
write_bytes	7.15E-02	1.84E-01	5.36E-01	8.45E-01	1.43E-02	9.43E-02	1.36E-02	1.06E-01	2.43E-02
rchar	3.49E-02	9.30E-02	1.34E-01	9.68E-01	1.18E-02	9.53E-02	2.87E-02	9.93E-02	2.88E-02
stime	3.15E-02	1.33E-01	6.87E-01	7.49E-01	1.11E-02	5.52E-01	1.06E+00	8.84E-02	1.63E-02
wchar	2.98E-02	9.52E-02	5.22E-01	6.90E-01	1.09E-02	9.97E-02	3.29E-02	8.35E-02	9.87E-03
utime	2.91E-02	9.33E-02	7.95E-01	6.75E-01	1.66E-02	8.30E-02	1.08E-02	8.90E-02	1.14E-02
rss	3.09E-02	8.09E-02	7.40E-01	6.53E-01	1.08E-02	4.49E-01	7.53E-01	9.78E-01	6.57E-01
received_bytes	2.83E-02	8.23E-02	6.52E-01	6.04E-01	1.07E-02	8.18E-02	9.15E-03	8.69E-02	1.16E-02
sent_bytes	2.97E-02	8.49E-02	4.28E-01	6.14E-01	1.06E-02	7.98E-02	7.18E-03	8.77E-02	1.09E-02

As far as average execution times are concerned, the results shown in Table 6 are very similar to those shown in Table 4, the RBFN model outperforms the other methods, being the fastest one in the calculation of the missing values for the 8 parameters that have been regressed. On the other hand, those methods based on regression trees (FT and BE) are the ones with highest execution times for all the components. Although more trees are generated in the case of the BE, their execution times are higher that those of the FT only for 5 of the attributes.

Additionally, it is noted that the execution times are somewhat smaller than those shown in Table 4 because the the number of predictors (attributes) for this component (7) is smaller than those for the ArmController (10).

In order to ease a visual analysis of these obtained results, Figure 4 and Figure 5 show the boxplots for the values that are summarized in Table 5 and Table 6 respectively.

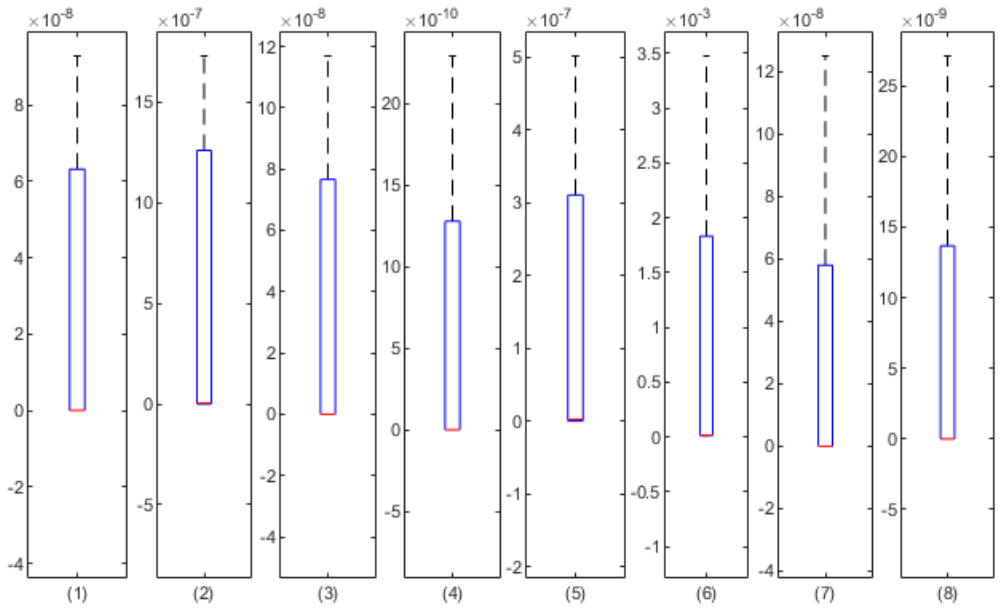

Figure 4. Boxplot of the MSE values (all imputation methods) on the LegDetector component per attribute. (1) write_bytes, (2) rchar, (3) stime, (4) wchar, (5) utime, (6) rss, (7) received_bytes, (8) sent_bytes

Figure 4 shows more compact results than those associated to the ArmController component (shown in Figure 2). In the case of the LegDetector component, there is no outliers in any of the attributes and all the error rates for the 8 attributes are within the 25^{th} and 75^{th} percentiles. This is a fact that deserves attention, as it means that the imputation methods perform in a regular and smooth way for all the attributes.

On the other hand, when considering the boxplot of the execution time (Figure 5), it can be said that similar results are shown to those for the ArmController component (shown in Figure 3). Once again, in the case of N-LR and the MLP with the BR training algorithm, an outlier is identified outside the percentiles. For all the other methods it can be said that time results are similar to those of the previous component (ArmController).

All in all, it can be said that for the two components that are analyzed in present research, error rates vary according to the regression method that is applied. As the aim of present work is to validate which one of these methods outperforms the other ones in order to impute missing values, MSE and time results (shown in Tables 3 to 6) are summarized in Table 7.

By means of Table 7 it is possible to analyze at a glance the obtained results that are presented in this section. In nutshell, the methods that have obtained the best results in terms of MSE are RBFN and N-LR. N-LR outperforms the other ones for most of the attributes in the ArmController component while the same happens with RBFN in the case of the LegDetector component. For 4 attributes LR, N-LR and RBFN have obtained similar error rates, that are the lowest ones. In addition to this general perspective, it can be seen that for some of the attributes, MLR_BR and

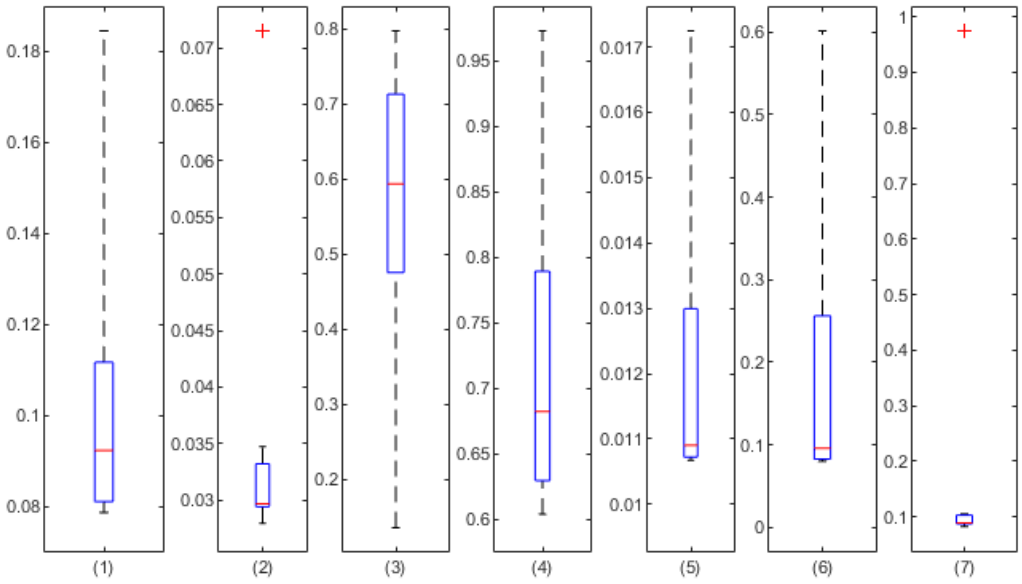

Figure 5. Boxplot of the execution time (all components) on the LegDetector component per method.(1) LR, (2) N-LR, (3) FT, (4) BE, (5) RBFN, (6) MLP_LM, (7) MLP_BR

Table 7. Summary of the best-performing imputation method per component attribute in terms of both error and execution time.

Dataset	ArmController		LegDetector	
Attribute	MSE	Time	MSE	Time
vsize	MLP_BR	RBFN	-	-
open_fds	FT	RBFN	-	-
open_connections	N-LR	RBFN	-	-
num_threads	N-LR	RBFN	-	-
rchar	N-LR	RBFN	LR, N-LR, RBFN	RBFN
stime	LR, N-LR, RBFN	RBFN	RBFN	RBFN
wchar	N-LR	RBFN	RBFN	RBFN
utime	N-LR	RBFN	RBFN	RBFN
rss	N-LR	RBFN	MLP_BR	RBFN
received_bytes	N-LR	RBFN	LR, N-LR, RBFN	RBFN
sent_bytes	N-LR	RBFN	RBFN	RBFN
write_bytes	-	-	LR, N-LR, RBFN	RBFN

FT are the best performing methods. This means that the selection of the regression method must be considered case by case and several methods must be applied in order to impute missing values with the minimum error. In terms of execution times, the RBFN model is the fastest one in all cases.

5. Conclusion

In the present study the imputation methods detailed in Section 2 have been applied to the two datasets explained in Section 3. These two datasets correspond to the ArmController and LegDetector components of the robot. After preparing the data and applying the CV scheme to obtain more reliable results, a regression has been performed on the 11 and 8 attributes of the dataset. From the obtained results (Section 4) it can be concluded that:

- For the ArmController component (Section 3), the N-LR method is the one that obtains the best results in terms of MSE and the neuronal model RBFN is the second best. The worst error rates are obtained by the FT and BE techniques. The attributes with lowest error rates have been open_fds and open_connections, so the imputation of missing values on them can be reliably performed. As for execution times, the RBFN method obtains the lowest times on all attributes.

- For the LegDetector component (Section 5), the RBFN method is the one that obtains the best results in terms of MSE for the 8 attributes. In many cases, similar results are obtained by LR and N-LR. Additionally, RBFN is the fastest method on all attributes.

- It can be observed that there is no single technique that is best in all cases. Even on the same attribute in the 2 different datasets, the best results can be obtained with different techniques.

Taking into account all the above mentioned, it can be concluded that imputation of missing values can be successfully performed. One of the regression methods that are compared can be selected to impute values of each one of the attributes from the given components.

As a future line of work, imputation will be combined with some other data preprocessing techniques (such as data balancing algorithms) to improve anomaly detection.

References

[1] G. Hoffman, Evaluating fluency in human–robot collaboration, IEEE Transactions on Human-Machine Systems 49 (3) (2019) 209–218 (June 2019). doi:10.1109/THMS.2019.2904558.
[2] E. Khalastchi, M. Kalech, On fault detection and diagnosis in robotic systems, ACM Comput. Surv. 51 (1) (2018) 1–24 (Jan. 2018). doi:10.1145/3146389.
[3] X. Xu, H. Liu, M. Yao, Recent progress of anomaly detection, Complexity 2019 (2019). doi:10.1155/2019/2686378.
[4] S. Ranshous, S. Shen, D. Koutra, S. Harenberg, C. Faloutsos, N. F. Samatova, Anomaly detection in dynamic networks: a survey, Wiley Interdisciplinary Reviews: Computational Statistics 7 (3) (2015) 223–247 (2015). doi:10.1002/wics.1347.
URL https://doi.org/20.1002/wics.1347
[5] E. Jove, J.-L. Casteleiro-Roca, H. Quintián, J. A. Méndez-Pérez, J. L. Calvo-Rolle, A fault detection system based on unsupervised techniques for industrial control loops, Expert Systems 36 (4) (2019) e12395 (2019). doi:10.1111/exsy.12395.
URL https://doi.org/10.1111/exsy.12395
[6] Á. Herrero, A. Jiménez, Improving the management of industrial and environmental enterprises by means of soft computing, Cybernetics and Systems 50 (1) (2019) 1–2 (2019).
[7] M. Dorigo, U. Schnepf, Genetics-based machine learning and behavior-based robotics: a new synthesis, IEEE Transactions on Systems, Man, and Cybernetics 23 (1) (1993) 141–154 (Jan 1993). doi:10.1109/21.214773.
[8] M. Jayaratne, D. de Silva, D. Alahakoon, Unsupervised machine learning based scalable fusion for active perception, IEEE Transactions on Automation Science and Engineering 16 (4) (2019) 1653–1663 (Oct 2019). doi:10.1109/TASE.2019.2910508.
URL https://doi.org/10.1109/TASE.2019.2910508
[9] J. Kober, J. A. Bagnell, J. Peters, Reinforcement learning in robotics: A survey, The International Journal of Robotics Research 32 (11) (2013) 1238–1274 (2013). doi:10.1177/0278364913495721.
URL https://doi.org/10.1177/0278364913495721
[10] D. Zhao, W. Ni, Q. Zhu, A framework of neural networks based consensus control for multiple robotic manipulators, Neurocomputing 140 (2014) 8 – 18 (2014). doi:10.1016/j.neucom.2014.03.041.
URL https://doi.org/10.1016/j.neucom.2014.03.041
[11] B. Xiao, S. Yin, Exponential tracking control of robotic manipulators with uncertain dynamics and kinematics, IEEE Transactions on Industrial Informatics 15 (2) (2019) 689–698 (Feb 2019). doi:10.1109/TII.2018.2809514.
[12] s. H. Alsamhi, O. Ma, M. S. Ansari, Survey on artificial intelligence based techniques for emerging robotic communication, Telecommunication Systems 72 (3) (2019) 483–503 (Nov 2019). doi:10.1007/s11235-019-00561-z.
URL https://doi.org/10.1007/s11235-019-00561-z
[13] H. Lu, Y. Li, S. Mu, D. Wang, H. Kim, S. Serikawa, Motor anomaly detection for unmanned aerial vehicles using reinforcement learning, IEEE Internet of Things Journal 5 (4) (2018) 2315–2322 (Aug 2018). doi:10.1109/JIOT.2017.2737479.
URL https://doi.org/10.1109/JIOT.2017.2737479

[14] J. Wienke, S. Meyer zu Borgsen, S. Wrede, A data set for fault detection research on component-based robotic systems, in: L. Alboul, D. Damian, J. M. Aitken (Eds.), Towards Autonomous Robotic Systems, Vol. 9716, Springer International Publishing, Cham, 2016, pp. 339–350 (2016).

[15] J. Wienke, S. Wrede, A Fault Detection Data Set for Performance Bugs in Component-Based Robotic Systems (2016). doi:10.4119/unibi/2900911.
URL https://doi.org/10.4119/unibi/2900911

[16] A. Arroyo, A. Herrero, V. Tricio, E. Corchado, M. Woźniak, Neural models for imputation of missing ozone data in air-quality datasets, Complexity 2018 (Mar 2018). doi:10.1155/2018/7238015.
URL https://doi.org/10.1155/2018/7238015

[17] B. Twala, Robot execution failure prediction using incomplete data, in: 2009 IEEE International Conference on Robotics and Biomimetics (ROBIO), 2009, pp. 1518–1523 (Dec 2009). doi:10.1109/ROBIO.2009.5420900.

[18] U. of Yale, Linear Regression (2017).
URL http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm

[19] U. of Yale, Multiple linear regression (2017).
URL http://www.stat.yale.edu/Courses/1997-98/101/linmult.htm

[20] G. G. Moisen, Classification and regression trees (2018).

[21] C. Lv, Y. Xing, J. Zhang, X. Na, Y. Li, T. Liu, D. Cao, F. Wang, Levenberg–marquardt backpropagation training of multilayer neural networks for state estimation of a safety-critical cyber-physical system, IEEE Transactions on Industrial Informatics 14 (8) (2018) 3436–3446 (Aug 2018). doi:10.1109/TII.2017.2777460.

[22] C. D. Doan, S.-y. Liong, Generalization for multilayer neural network bayesian regularization or early stopping, in: Proceedings of Asia Pacific Association of Hy drology and Water Resources 2nd Conference, 2004, pp. 5–8 (2004).

[23] F. Jumel, Advancing research at the robocup@home competition [competitions], IEEE Robotics Automation Magazine 26 (2) (2019) 7–9 (June 2019). doi:10.1109/MRA.2019.2908571.
URL https://doi.org/10.1109/MRA.2019.2908571

[24] J. Wienke, S. Wrede, A middleware for collaborative research in experimental robotics, in: 2011 IEEE/SICE International Symposium on System Integration (SII), 2011, pp. 1183–1190 (Dec 2011). doi:10.1109/SII.2011.6147617.
URL https://doi.org/10.1109/SII.2011.6147617

[25] J. Wienke, S. Wrede, Autonomous fault detection for performance bugs in component-based robotic systems, in: Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on, IEEE, 2016, pp. 3291–3297 (2016). doi:0.1109/IROS.2016.7759507.