Enhancing Auto-ML with Missing Value Imputation: A Case Study with TPOT2 Library and Industry 4.0

Joel Frank Huarayo Quispe[1] **, Didier A. Vega-Oliveros**[1] **, Lilian Berton**[1]

[1]Instituto de Ciência e Tecnologia – Universidade Federal de São Paulo (UNIFESP)
12247-014– São José dos Campos – SP – Brazil

{joel.frank,didier.vega,lberton}@unifesp.br

Abstract. *Automated Machine Learning (AutoML) is increasingly important in industrial applications for democratizing the use of machine learning techniques, particularly in Industry 4.0, where robust model development is crucial. Addressing the challenge of missing data, we introduce a missing data imputation module integrated into the TPOT2 AutoML library—a rewrite of TPOT with additional features. This module incorporates SimpleImputer, IterativeImputer, and KNNImputer, enhancing TPOT2's ability to handle datasets with missing values. We evaluate the module on three industrial datasets (Mercedes-Benz Greener Manufacturing, NASA Turbofan Jet Engine, Gearbox fault diagnosis) with classification and regression tasks, testing it with varying levels of missing data (5%, 10%, 15%). Our results demonstrate that the TPOT2 library, equipped with this imputation module, significantly improves predictive modeling accuracy in the presence of missing data, proving its practical utility and robustness in industrial contexts.*

1. Introduction

Automated Machine Learning (AutoML) is revolutionizing the accessibility and application of machine learning by automating the entire model development process [He et al. 2021], including algorithm selection, hyperparameter tuning, and feature engineering. This automation lowers the entry barriers for general users and enhances efficiency across diverse applications, allowing users to focus on interpreting and applying results rather than managing complex technical details. Particularly in the context of Industry 4.0, AutoML enables the rapid deployment of predictive models in manufacturing and industrial processes [Jan et al. 2023], allowing industries to effectively leverage data-driven decision-making.

While AutoML has demonstrated significant advancements in simplifying the ML workflow, it is not without limitations, particularly in the realm of preprocessing [Bilal et al. 2022, Shende et al. 2022]. AutoML tools often face challenges in handling highly customized or domain-specific preprocessing requirements. The automated nature of these platforms may result in less flexibility for users who need to incorporate domain knowledge or specific data transformations tailored to their problem. Additionally, AutoML tools might struggle with data preprocessing tasks involving complex feature engineering or handling missing values in a manner that aligns precisely with the nuances of a particular dataset. Striking a balance between automation and user customization remains an ongoing challenge, as the one-size-fits-all approach inherent in many AutoML solutions may not cater to the intricacies of every unique dataset or domain.

Missing value imputation is a crucial process in data preprocessing that addresses the challenge of incomplete data by filling in absent values with estimated or predicted values [Lin and Tsai 2020]. In various real-world datasets, missing values can arise due to multiple factors, such as data collection errors, equipment malfunctions, or intentional omissions. The goal of imputation is to enhance the completeness and reliability of the dataset, allowing for more robust analysis and modeling. Numerous techniques are employed for imputing missing values, including statistical methods, machine learning algorithms, and domain-specific approaches [Lin and Tsai 2020]. Each method has its strengths and limitations, and the choice of imputation strategy often depends on the nature of the dataset and the specific goals of the analysis. Effective missing value imputation contributes to more accurate and reliable results in subsequent data analysis and machine learning tasks.

Within the research on ML automation, several optimization techniques have been developed to find the best parameter setting that requires less effort for the data scientist and fewer expert people. For this research, we chose the concept of tree-based pipeline optimization, focusing more specifically on the TPOT2 [1] tool (which is a rewrite of TPOT [Olson et al. 2016] found in open source alpha version). TPOT2 makes it easier for others to develop and improve the algorithm, whose purpose is to help optimize the most tedious part of ML through genetic programming [Banzhaf et al. 1998], where all pipe operators make use of existing implementations in Scikit-learn [Pedregosa et al. 2011]. Unlike TPOT, this tool has additional functions and parameters, like generating pipelines in graphics to better specify the search space in each pipeline operator.

In this work, a missing value imputation module was developed as a case study for integration into the AutoML library TPOT2. This module[2] serves as a crucial enhancement, addressing the common challenge of missing data in real-world scenarios. Testing was conducted on three distinct datasets, each focusing on fault detection in industrial equipment within the industrial context. The evaluation spanned varying percentages of missing values (5%, 10%, and 15%) to comprehensively assess the module's efficacy under different conditions.

2. Related Work

Few works have developed approaches for preprocessing in AutoML. [Bilal et al. 2022] developed a Python-based Auto-preprocessing architecture for AutoML to recommend the most effective data cleaning and preparation method to the user after evaluating prior-art candidate techniques. `cleanTS` [Shende et al. 2022] automates the preprocessing of univariate time series data to improve efficiency in the cleaning process and reduce data preparation time. Implemented as an R package, it also enhances the visualization of large datasets, supporting data analysis at various scales and resolutions. `HyperImpute` was proposed by [Jarrett et al. 2022], as an iterative imputation framework that combines the simplicity and customizability of conventional approaches with the efficiency of deep generative modeling. The proposed framework automatically configures column-wise models and their hyperparameters, offering a practical implementation with various components. [Bilalli et al. 2016] developed an automated approach using meta-learning concepts, considering multiple data pre-processing techniques and data mining algorithms.

[1]`https://epistasislab.github.io/tpot2/`, accessed on August 28, 2024

[2]The code is available at `https://github.com/Prescriptive-Maintenance-IAsmin`

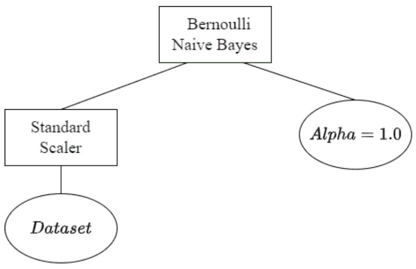


Figure 1. A visual representation of a tree-based ML pipeline where squares represent primitives and ellipses are terminals. This example applies standard scaling to the data before learning a Bernoulli Naive Bayes model.

Their method predicts the transformations that enhance algorithm performance on specific datasets. [Torniainen et al. 2020] introduce `Nippy`, an open-source Python module designed for semi-automatic comparison of Near-infrared spectroscopy (NIRS) preprocessing methods. This tool streamlines the selection process, enhancing the optimization of NIRS models. As far as we know, no previous work proposed a missing value imputation module integrated with TPOT framework, as presented here.

Several reviews explore preprocessing methods across different domains. [García et al. 2016] discusses data preprocessing techniques in big data, focusing on frameworks like Hadoop, Spark, and Flink. [Chai 2023] examines the advantages and disadvantages of common text preprocessing methods, including tokenization, text normalization, and n-gramming. [Mishra et al. 2020] reviews recent developments in ensemble preprocessing strategies in chemometrics, highlighting their applicability in improving model performance. Finally, [Alghamdi and Javaid 2022] surveys data preprocessing methods in the smart grid domain, emphasizing their importance for accurate forecasting in electricity demand, generation, and pricing.

3. Methods

3.1. TPOT2

The Tree-based Pipeline Optimization Tool (TPOT2) is a robust and user-friendly tool, designed to automate the selection of optimal ML pipelines, including preprocessing, model selection, and hyperparameter tuning [Olson et al. 2016]. It leverages tree-based genetic programming, where the tree consists of nodes—primitives requiring input data, hyperparameters, and terminals, as constants passed to the primitives (Figure 1).

In TPOT2, channels within the genetic algorithm serve as entities that facilitate operations like mutation—modifying hyperparameters or adding preprocessing steps—and crossover, where primitives interact between pipelines to exchange subtrees or leaves. These pipelines are then evaluated and assigned a fitness score, determining which individuals proceed to the next generation. This process efficiently explores several possibilities, optimizing the pipeline for the given dataset. TPOT2 stands out for its ability to manage end-to-end ML workflows with minimal user intervention, making it accessible to users with varying experience levels. Its adaptability, versatility, and automation capabilities position it as a valuable tool in the evolving field of AutoML.

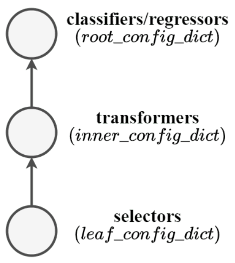


Figure 2. The $root\_config\_dict$ specifies the modules for the root node: a classifier, regression module, or a transformer. The $inner\_config\_dict$ specifies the modules allowed in all non-root nodes. If the leaf dictionary is set to `None`, leaves will be selected from this list (transformers), but the presence of a node from this list is not guaranteed, potentially resulting in a graph with only a root or a root and a leaf. The $leaf\_config\_dict$ defines the modules that can be used as leaves, and unlike $inner\_config\_dict$, it guarantees the presence of a leaf node if specified. Thus, the smallest possible network would be $[leaf \rightarrow root]$.

3.1.1. TPOT2 Overview

There are two different evolutionary algorithms integrated into TPOT2, which correspond to two different estimators, these will be described below:

- *TPOTEstimator*: TPOT2 employs a standard evolutionary algorithm, evaluating all individuals in the population for each generation sequentially. A new generation begins only after the previous one has been fully evaluated, which leads to a larger computational time but helps to preserve population diversity.
- *TPOTEstimatorSteadyState*: The way it works is how soon each individual finishes evaluating itself, and the next one begins to be generated and evaluated successively. This allows for more efficient utilization when using cores.

Both algorithms have a simplified set of hyperparameters with default values set for classification and regression problems as roots of the tree (this differs from TPOT because classifiers and regressors can appear in locations other than the root). TPOT2 also addresses the design of overly complex pipelines by integrating *Pareto* optimization, which produces compact pipelines without sacrificing classification/regression accuracy.

3.1.2. Defining Search Space

Here, we worked with the *TPOTEstimator* class. We focused on the search spaces provided by the class for each of the nodes. TPOT2 will generate pipelines with a default set of classifiers and regressors as roots (depending on the assigned configuration), all other nodes are selected from a default list of selectors and transformers. It is possible to modify the search space of leaves, internal nodes, and roots separately using built-in options or custom configuration dictionaries. In this work, we focus on the creation of custom configurations and the use of nested configurations, which will be explained below.

TPOT2 consists of three different configuration dictionaries to indicate the type of modules that should be in the graph node, as shown in Figure 2. We propose to include a new dictionary called *data_cleaning* as part of the tree, as shown in Figure 3. The new *data_cleaning* search space consists of three configuration dictionaries nested as pipeline

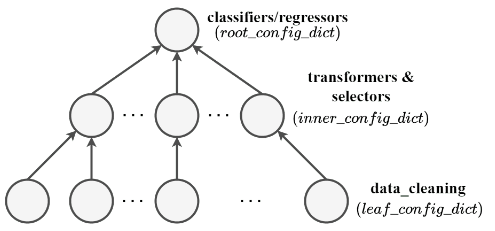


Figure 3. This graph shows the result after adding data cleaning as another search space named: $data\_cleaning$, focusing on data imputation, and this new search space would now become the $leaf\_config\_dict$ of the tree; the transformers and selectors become the $inner\_config\_dict$ search space and the $root\_config\_dict$ according to the problem can be classifiers or regressors.

leaves, within each dictionary an Optuna [Akiba et al. 2019] compatible function is created, which performs a combined test of preset hyperparameters according to the function and returns a dictionary of hyperparameters.

3.2. Reference data sets

To demonstrate the efficiency of the new search space that contains data imputation methods, three data sets oriented to industry 4.0 have been considered, which have to do with production efficiency and failure predictions in industrial equipment, these data are obtained in competitions within the *Kaggle* platform.

Mercedes-Benz Greener Manufacturing [3]: The data consists of 4,209 data samples from the car testing system before hitting the road. The aim is to optimize the speed of your testing system since it has many functions to test by combining them. This data set contains an anonymous set of variables, each of which represents a personalized feature on a Mercedes car. The target characteristic is labeled y and represents the time (in seconds) it took the car to pass the test for each variable (variables with letters are categorical and variables with 0/1 are binary values).

NASA Turbofan Jet Engine Data Set [4]: The data consists of 350,727 data samples from NASA asset degradation modeling. Engine degradation simulation was carried out using *C-MAPSS* ('Commercial Modular Aerodynamics Propulsion System Simulation' and is a tool for realistic data simulation of large commercial turbofan engines). Four different assemblies were simulated under different combinations of operating conditions and failure modes. Records multiple sensor channels (17 sensors) to characterize fault evolution.

Gearbox fault diagnos[5]: The data consists of 8,084,476 data samples from gearbox fault diagnosis, which includes the vibration data set recorded by using the gearbox fault diagnosis simulator called *SpectraQuest*. The data set was recorded using 4 vibration sensors placed in four different directions and under a load variation from '0' to '90' percent. Two different scenarios are included (healthy condition and broken tooth condition).

[3]https://kaggle.com/competitions/mercedes-benz-greener-manufacturing
[4]https://www.kaggle.com/datasets/behrad3d/nasa-cmaps
[5]ishttps://www.kaggle.com/datasets/brjapon/gearbox-fault-diagnosis

3.3. Missing value imputation

Data preprocessing is a crucial yet time-consuming aspect of ML automation, particularly concerning the imputation of missing values. It is considered essential for preparing data for tasks like classification, forecasting, and clustering [Lakshminarayan et al. 1999]. While functions can be created to automate imputation, selecting the most suitable method for a specific model requires significant time and effort. If the missing data length is extensive, the uncertainty increases, making accurate imputation more challenging; in such cases, it may be preferable to eliminate periods with missing data [Gourraud et al. 2004]. The decision on what constitutes "short" or "long" missing data depends on the data's representation and its intended use.

In our study on evolutionary algorithms, we plan to incorporate popular data-cleaning techniques into the search space, allowing researchers to experiment with these methods and create new ML pipelines. Specifically, we focus on integrating data imputation algorithms using three general methods available in the Scikit-learn library.

- *SimpleImputer [Jackson et al. 1982]:* Replace missing values using a descriptive statistic (e.g. mean, median, or most frequent) along each column, or using a constant value.
- *IterativeImputer [Little and Rubin 1986]:* Multivariate computer that estimates each feature from all the others. The strategy is to impute missing values by modeling each feature with missing values based on other features in a circular fashion.
- *KNNImputer [Troyanskaya et al. 2001]:* Imputation to fill missing values using k-nearest neighbors. Missing values for each sample are imputed using the mean value of $n\_neighbors$ nearest neighbors found in the training set. Two samples are close if the characteristics that neither of them is missing are close.

3.4. Generation of missing data

To ensure that a consistent percentage of missing values is present in the three data sets mentioned in Section 3.2, we artificially and randomly generate these missing values, as described below.

- *Regression:* For regression problems, we obtain two data sets: Mercedes-Benz Greener Manufacturing and NASA Turbofan Jet Engine Data Set. For the Mercedes-Benz data set, when imputing data, we considered missing data in small intervals and also in longer periods. To simulate this case, missing values were generated from the total of values that were 4,209 data samples, 5% (represents 210 removed values), 10% (represents 421 removed values), and 15% (represents 631 removed values). These data were removed from 4 different sectors, creating a total of 3 new data sets with missing values. We removed data from the objective function y. The MICE [Zhang 2016] method was used, which focuses on imputing data iteratively using a series of regression models.
  The NASA Turbofan data set has data from several sensors that can determine the lifetime of the turbofan. To do so, we attempted to randomly eliminate the values of the readings from some of the 17 sensors. In the 17 sensors considered there is a total of 350,727 registers, imputing 4.88% (represents 17,124 removed values), 9.53% (represents 33,432 removed values), and 13.87% (represents 48,652 removed values).

Table 1. Regression metrics obtained from running with TPOT2 on the Mercedez-Benz dataset with full values and data imputation at 5%, 10%, 15%, respectively.

	0%NaN	5%NaN	10%NaN	15%NaN
r2_score Pareto	0.564203	0.530479	0.496949	0.463484
MSE	54.293988	55.946215	49.489081	52.854175
RMSE	7.368445	7.479720	7.034847	7.270087
R-squared	0.619054	0.564966	0.588331	0.532672
MAE	5.209274	5.492053	5.146635	5.469484

- *Classification:* The GearBox data set contains the reading of 4 sensors over time to determine if the state of the gearbox is broken or healthy. Some data were randomly removed from the total reading of the 4 sensors, which corresponds to 8,084,476 data. The elimination percentage was as follows: 4.88% (represents 394,384 removed values), 9.51% (represents 768,986 removed values), and 13.93% (represents 1,125,896 removed values).

4. Results

TPOT2 was tested with the new dataset generated with null values and the new *data_cleaning* search space that contains the data imputation dictionaries, the automatic optimization of the hyperparameters designed by Optuna, and the pipeline optimization based on TPOT2 tool trees.

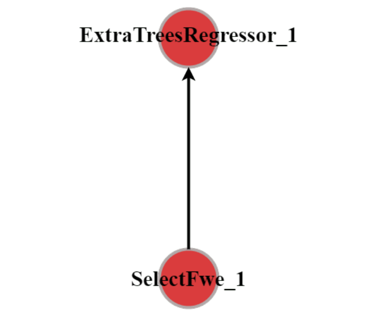
4.1. Mercedes-Benz Greener Manufacturing

This data set has two scripts, one for training and one for testing (this one does not have the objective function y, which is intended to predict and know the scores). Within the configuration of the TPOT2 tool, it is necessary to specify the objective function (y_train) and the columns of the other variables (X_train) in regression problems. Therefore the search spaces only affect the columns of X_train and not in y_train. The data imputation was previously performed with *IterativeImputer*, as well as the categorical variables were preprocessed with *MultiLabelBinarizer* and eliminated the constant variables that do not affect the objective function, finally, it was passed through TPOT2 to generate the pipelines (Figure 4) as well as to know the scores of the evaluation metrics.

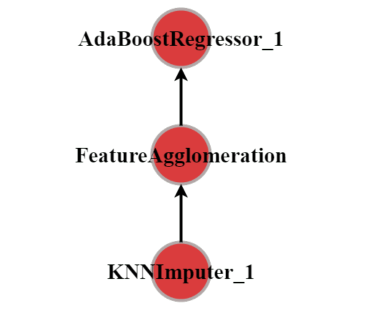
In Table 1, the 0%NaN column means that the y_train data was considered without data extraction and without the data imputation search space. The other percentages indicate the dataset has missing data and passed it to the data imputation module.

4.2. NASA Turbofan Jet Engine Data Set

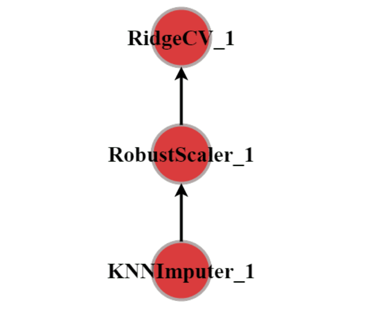
This dataset has multivariate time series data, whose objective function is RUL (Remaining Useful Life) prediction of the turbofan engine, which means the maximum cycle minus the cycle in each stage for each engine. The goal is to predict the number of operating cycles remaining before failing on the test set. For this test, data from a single simulation "FD001" was used in conditions: ONE (Sea Level) and failure mode: ONE (HPC Degradation). This dataset does not contain readings with the columns of sensors 26 and 27, therefore they are considered as *NaN* values which will not be useful for our case study. It also has 7 columns with constant values that will be removed because they do not affect the objective function RUL. Data elimination was carried out randomly in the readings



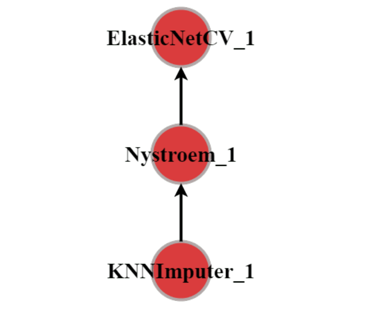
(a) Pipeline generated with 0% imputed data and without the data_cleaning search space



(b) Pipeline generated with 5% imputed data and nested with the data_cleaning search space



(c) Pipeline generated with 10% imputed data and nested with the data_cleaning search space



(d) Pipeline generated with 15% imputed data and nested with the data_cleaning search space

Figure 4. The pipelines generated by TPOT2 after training with the regression-type data Mercedez-Benz.

of the different sensors because they are numerical data, and it is possible to apply data imputation algorithms.

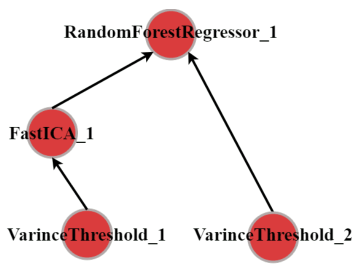
Table 2 shows the results of our case study. When running with 0% data, the default TPOT2 configuration was used without the *data_cleaning* search space to have a reference with the results of the datasets that have 4.88%, 9.53%, 13.87% of deleted data. Likewise, Figure 5 shows the different pipelines that were generated by TPOT2.

4.3. Gearbox fault diagnosis

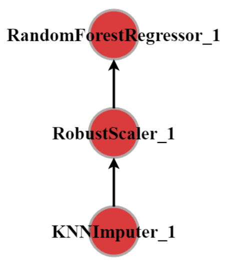
This data set contains simulated data of gearbox failures, being of binary classification type, where 4 sensors are recorded in 10 different scenarios (each scenario has a load from 0% to 90%). These readings do not have empty data, so some readings at 4.88%, 9.51%, and 13.93% in different sensors had to be eliminated for data imputation. Table

Table 2. Regression metrics obtained from running with TPOT2 on the Nasa-Turbofan dataset with full values and different data imputation percentages.

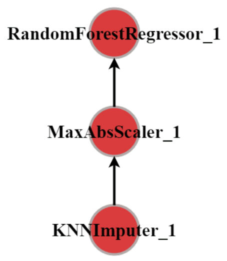
	0%NaN	4.88%NaN	9.53%NaN	13.87%NaN
r2_score Pareto	0.9059	0.965054	0.865393	0.683005
MSE	243.005165	6.421258	382.135715	1553.697680
RMSE	15.588622	2.534020	19.548291	39.416971
R-squared	0.949313	0.998660	0.920294	0.675929
MAE	11.050225	1.381818	14.032818	29.049405



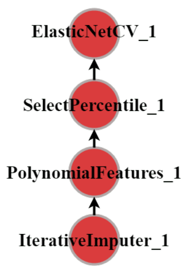
(a) Pipeline generated with 0% imputed data and without the data_cleaning search space



(b) Pipeline generated with 5% imputed data and nested with the data_cleaning search space



(c) Pipeline generated with 10% imputed data and nested with the data_cleaning search space



(d) Pipeline generated with 15% imputed data and nested with the data_cleaning search space

Figure 5. The pipelines generated by TPOT2 after training with the regression-type Nasa-Turbofan data.

3 shows the results, where 0% of imputed data does not have the *data_cleaning* search space when executing TPOT2. The results indicate that the pipeline complexity increases as the percentage of imputed data rises compared to the original dataset. On the contrary, when eliminating the data at different fractions, we have the TPOT2 configuration with *data_cleaning* for data imputation. Likewise, Figure 4.3 shows the different pipelines generated by TPOT2.

5. Discussion and Final Remarks

This paper presents the implementation of three imputation approaches for different cases using TPOT2 for tree-based pipeline optimization, aiming to minimize data preprocessing for data scientists. The Pareto metric was employed to design compact, interpretable pipelines without compromising regression or classification accuracy. Our results show that by applying the *SimpleImputer*, *IterativeImputer*, and *KNNImputer* algorithms, it is possible to achieve results comparable to those obtained with complete datasets.

We showed that integrating new search spaces can significantly aid machine learning professionals in selecting the best algorithms and parameters for their data. In this case study, we introduced a new search space called “*data_cleaning*”, specifically tested on Industry 4.0 problems related to sensor data, motor maintenance, and productivity improvement over time. Our tests focused on regression and classification tasks. TPOT2 does not include data imputation in its framework, making it necessary to manually specify the type of machine learning problem and apply the appropriate data imputation algorithm.

In the Mercedes-Benz dataset, a time series regression problem, we generated missing data and considered removing segments of the objective function to allow data imputation using the *IterativeImputer*, which references previous data. Additionally, the

Table 3. Regression metrics from TPOT2 on the Gearbox fault diagnosis dataset evaluated with full data and data imputation at 4.88%, 9.51%, and 13.93%. Validation data were randomly obtained with consistent amounts and patterns.

0%NaN

roc_auc Pareto	0.925329			
	precision	recall	f1-score	support
broken(0)	1.00	1.00	1.00	805
healthy(1)	1.00	1.00	1.00	803
accuracy			1.00	1608
macro avg	1.00	1.00	1.00	1608
weighted avg	1.00	1.00	1.00	1608

4.88%NaN

roc_auc Pareto	0.525584			
	precision	recall	f1-score	support
broken(0)	0.50	0.69	0.58	805
healthy(1)	0.50	0.32	0.39	803
accuracy			0.50	1608
macro avg	0.50	0.50	0.49	1608
weighted avg	0.50	0.50	0.49	1608

9.51%NaN

roc_auc Pareto	0.519652			
	precision	recall	f1-score	support
broken(0)	0.55	0.41	0.47	805
healthy(1)	0.53	0.66	0.58	803
accuracy			0.53	1608
macro avg	0.54	0.53	0.53	1608
weighted avg	0.54	0.53	0.53	1608

13.93%NaN

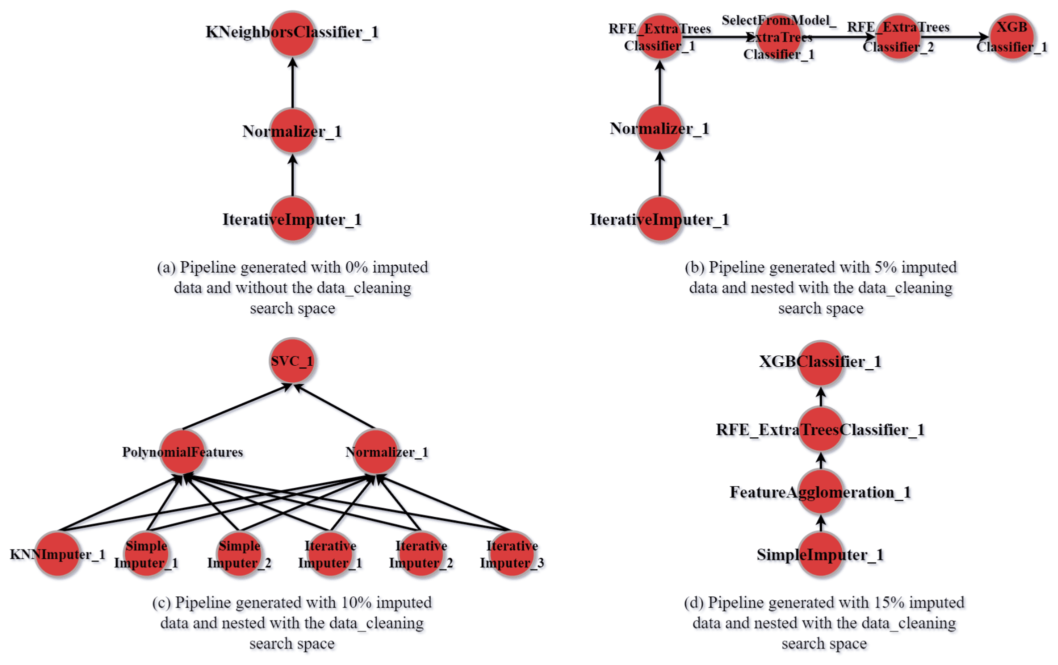
roc_auc Pareto	0.525583			
	precision	recall	f1-score	support
broken(0)	0.52	0.62	0.56	805
healthy(1)	0.52	0.42	0.47	803
accuracy			0.52	1608
macro avg	0.52	0.52	0.51	1608
weighted avg	0.52	0.52	0.51	1608

MultiLabelBinarizer function was applied to convert categorical values into binary numerical values. Although the *IterativeImputer* is included in the *data_cleaning* search space, TPOT2 does not directly apply it to the objective function, so this step was performed manually for analysis.

In future work, we can investigate the performance of additional data imputation algorithms, evaluate diverse datasets, address computational efficiency as new modules are included in the AutoML pipeline, and study the impact on other downstream tasks.

6. Acknowledgments

This work has been supported by Fundação de Amparo a Pesquisa do Estado de São Paulo (FAPESP) Grant 20/09850-0, Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).



(a) Pipeline generated with 0% imputed data and without the data_cleaning search space

(b) Pipeline generated with 5% imputed data and nested with the data_cleaning search space

(c) Pipeline generated with 10% imputed data and nested with the data_cleaning search space

(d) Pipeline generated with 15% imputed data and nested with the data_cleaning search space

Figure 6. The pipelines generated by TPOT2 after training with the classification-type Gearbox data.

References

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631.

Alghamdi, T. A. and Javaid, N. (2022). A survey of preprocessing methods used for analysis of big data originated from smart grids. *IEEE Access*, 10:29149–29171.

Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998). *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers Inc.

Bilal, M., Ali, G., Iqbal, M. W., Anwar, M., Malik, M. S. A., and Kadir, R. A. (2022). Auto-prep: efficient and automated data preprocessing pipeline. *IEEE Access*, 10:107764–107784.

Bilalli, B., Abelló, A., Aluja-Banet, T., and Wrembel, R. (2016). Automated data pre-processing via meta-learning. In *International Conference on Model and Data Engineering*, pages 194–208. Springer.

Chai, C. P. (2023). Comparison of text preprocessing methods. *Natural Language Engineering*, 29(3):509–553.

García, S., Ramírez-Gallego, S., Luengo, J., Benítez, J. M., and Herrera, F. (2016). Big data preprocessing: methods and prospects. *Big data analytics*, 1:1–22.

Gourraud, P., Ginin, E., and Cambon-Thomsen, A. (2004). Handling missing values in population data: Consequences for maximum likelihood estimation of haplotype frequencies. *European Journal of Human Genetics*, 12(10):805–812.

He, X., Zhao, K., and Chu, X. (2021). Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622.

Jackson, W., McNee, R., and TX., S. O. A. M. B. A. (1982). *An Algorithm for the Univariate Analysis of Variance in Experiments with Repeated Measures*. Defense Technical Information Center.

Jan, Z., Ahamed, F., Mayer, W., Patel, N., Grossmann, G., Stumptner, M., and Kuusk, A. (2023). Artificial intelligence for industry 4.0: Systematic review of applications, challenges, and opportunities. *Expert Systems with Applications*, 216:119456.

Jarrett, D., Cebere, B. C., Liu, T., Curth, A., and van der Schaar, M. (2022). Hyperimpute: Generalized iterative imputation with automatic model selection. In *International Conference on Machine Learning*, pages 9916–9937. PMLR.

Lakshminarayan, K., Harp, S. A., and Samad, T. (1999). Imputation of missing data in industrial databases. *Applied intelligence*, 11(3):259–275.

Lin, W.-C. and Tsai, C.-F. (2020). Missing value imputation: a review and analysis of the literature (2006–2017). *Artificial Intelligence Review*, 53:1487–1509.

Little, R. J. A. and Rubin, D. B. (1986). *Statistical Analysis with Missing Data*. John Wiley & Sons, Inc., USA.

Mishra, P., Biancolillo, A., Roger, J. M., Marini, F., and Rutledge, D. N. (2020). New data preprocessing trends based on ensemble of multiple preprocessing techniques. *TrAC Trends in Analytical Chemistry*, 132:116045.

Olson, R. S., Bartley, N., Urbanowicz, R. J., and Moore, J. H. (2016). Evaluation of a tree-based pipeline optimization tool for automating data science. *CoRR*, abs/1603.06212.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830.

Shende, M. K., Feijoo-Lorenzo, A. E., and Bokde, N. D. (2022). cleants: Automated (automl) tool to clean univariate time series at microscales. *Neurocomputing*, 500:155–176.

Torniainen, J., Afara, I. O., Prakash, M., Sarin, J. K., Stenroth, L., and Töyräs, J. (2020). Open-source python module for automated preprocessing of near infrared spectroscopic data. *Analytica Chimica Acta*, 1108:1–9.

Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P., Hastie, T., Tibshirani, R., Botstein, D., and Altman, R. B. (2001). Missing value estimation methods for DNA microarrays . *Bioinformatics*, 17(6):520–525.

Zhang, Z. (2016). Multiple imputation with multivariate imputation by chained equation (mice) package. *Annals of translational medicine*, 4(2).