Classification of Failure Using Decision Trees Induced by Genetic Programming

ARTICLE HISTORY

Received 2 January 2024
Accepted 19 April 2024

Rogério Costa Negro Rocha
Graduate Program in Computational Modeling and Systems
State University of Montes Claros
Montes Claros, Brazil
rogeriocostanegro@hotmail.com
ORCID: 0009-0002-4667-9656

Laércio Ives Santos
Campus Montes Claros Federal Institute of Norte de Minas Gerais
Montes Claros, Brazil
laercio.ives@gmail.com
ORCID: 0000-0001-6504-7692

Rafael Almeida Soares
Graduate Program in Computational Modeling and Systems
State University of Montes Claros
Montes Claros, Brazil
rafael.almeida.soares2012@gmail.com
ORCID: 0009-0006-5544-9798

Franciele Alves Barbosa
Graduate Program in Computational Modeling and Systems
State University of Montes Claros
Montes Claros, Brazil
francielealvesb10@gmail.com
ORCID: 0009-0005-3964-7391

Marcos Flávio Silveira Vasconcelos D'Angelo
Departament of Computer Science State University of Montes Claros
Montes Claros, Brazil
marcos.dangelo@unimontes.br
ORCID: 0000-0001-5754-3397

Classification of Failure Using Decision Trees Induced by Genetic Programming

1st Rogério Costa Negro Rocha
Graduate Program in Computational Modeling and Systems
State University of Montes Claros
Montes Claros, Brazil
rogeriocostanegro@hotmail.com
ORCID: 0009-0002-4667-9656

2nd Laércio Ives Santos
Campus Montes Claros
Federal Institute of Norte de Minas Gerais
Montes Claros, Brazil
laercio.ives@gmail.com
ORCID: 0000-0001-6504-7692

3rd Rafael Almeida Soares
Graduate Program in Computational Modeling and Systems
State University of Montes Claros
Montes Claros, Brazil
rafael.almeida.soares2012@gmail.com
ORCID: 0009-0006-5544-9798

4th Franciele Alves Barbosa
Graduate Program in Computational Modeling and Systems
State University of Montes Claros
Montes Claros, Brazil
francielealvesb10@gmail.com
ORCID: 0009-0005-3964-7391

5th Marcos Flávio Silveira Vasconcelos D'Angelo
Departament of Computer Science
State University of Montes Claros
Montes Claros, Brazil
marcos.dangelo@unimontes.br
ORCID: 0000-0001-5754-3397

***Abstract*—Fault classification in industrial processes is of paramount importance, as it allows the implementation of preventive and corrective measures before catastrophic failures occur, which can result in significant repair costs and production loss, for example. Therefore, the purpose of this study was to develop a classification model by merging the concepts of Decision Trees with Genetic Programming. To accomplish this, the proposed model randomly generates a set of decision trees using the adapted Tennessee Eastman dataset. The generation of these trees does not rely on classical construction logic; instead, they employ an approach where the structure and characteristics of the trees are randomly determined and adjusted throughout the evolutionary process. This approach enables a broader exploration of the search space and may lead to diverse solutions. The results obtained were moderate, largely due to the high number of target classes for classification (21 classes), resulting in the creation of complex trees. The average accuracy on the test data was 0.75, indicating the need to implement new alternatives and enhancements in the algorithm to improve the results.

Keywords*—*decision trees, multiclass classification, fault detection, genetic programming

I. Introduction

Fault Classification Problems have been widely explored in various fields of engineering and science, being of utmost relevance for the detection and prevention of anomalies in systems and processes. Early fault detection plays a fundamental role in proactive maintenance and ensuring efficient and safe operations in complex and interconnected environments, such as industrial production systems, telecommunications networks, and transportation systems.

In this context, fault classification involves the development of models and algorithms capable of identifying subtle patterns in data, distinguishing between normal situations and anomalous behaviors. Advanced machine learning and data mining techniques have been applied to address these challenges.

Precise fault detection and classification not only reduce costs associated with unplanned downtime, but also contribute to process optimization and worker safety.

To achieve this, models based on Machine Learning (ML) techniques can be used for fault classification. ML techniques are interesting because they emulate the human way of thinking and decision-making, analyze large datasets containing many features in a reasonable time, and can handle complex relationships between data, making them more accurate than human experts in some specific situations [1].

Given the above, decision trees are considered, a ML technique used for classification problems. The use of decision trees has proven to be a promising approach in the context of fault classification problems. Decision trees offer an intuitive and interpretable way to model complex patterns in data, allowing the identification of relevant features for the classification of different types of faults in industrial systems. Additionally, as highlighted by [2], the hierarchical nature of decision trees mirrors human decision-making processes, making them suitable for analyzing anomalous behaviors in interconnected systems.

The application of decision trees for fault classification can also be enhanced with data preprocessing techniques, such as relevant feature selection. Thus, the use of decision trees offers a versatile and effective approach to address the challenges of fault detection and classification in complex systems. However, such algorithms often use a greedy strategy and tend to fall into local optima. Moreover, the recursive partitioning policy in the construction phase can result in datasets with low cardinality for the attribute selection process in deeper tree nodes, causing data overfitting.

Furthermore, researchers have considered the application of evolutionary algorithms to induce decision trees, specifically through Genetic Programming (GP). GP is an evolutionary algorithm that evolves a set of individuals represented in the form of trees [3]. When GPs are applied to the induction of decision trees, it is possible to deal with multiple attributes simultaneously, reducing the dependence on feature selection methods in preprocessing and still providing a global search strategy [4]. This is an interesting approach to be tested, given that in recent years, it is not common to find works that use evolutionary computation techniques to induce decision trees.

Therefore, this work aims to build a multiclass classification algorithm based on decision trees induced by genetic programming, with the purpose of classifying faults in the adapted database of the Tennessee Eastman Process Simulation and analyzing its accuracy results. Experiments were conducted to assess the quality and complexity of the solutions found. The results obtained indicated that the model presents moderate results for fault classification in the chosen database and results in complex trees; therefore, new strategies need to be applied to the algorithm to achieve better results and performance.

II. Literature Review

A. Decision Trees

Decision Trees are widely used algorithms in machine learning to solve classification and regression problems. Data is organized in a tree-like structure, wherein each inner node signifies a decision derived from a particular attribute, and each terminal node, or leaf, corresponds to either a classification label or a regression value [5].

One of the advantages of decision trees is their interpretability. Their representation, especially when viewed graphically, is easily understandable. One can follow the logic of each node and interpret it until reaching a leaf node, which indicates the class of the instance, for example. Additionally, decision trees have the ability to handle both numerical and categorical data. They can represent complex relationships between attributes and classes, making them suitable for modeling nonlinear data [6].

To evaluate a decision tree, the Misclassification Error criterion can be used [6]. In this criterion, the number of correct predictions is measured by comparing predicted outputs with true outputs, resulting in accuracy. Accuracy assesses the ratio of correctly classified examples to the total number of evaluated examples. Higher accuracy indicates a greater number of accurate predictions.

B. Genetic Programming

Genetic Programming (GP) is an artificial intelligence technique that uses principles inspired by biological evolution to evolve solutions for complex problems. In this approach, a set of random solutions is represented as genetic structures that can be combined and mutated over several generations, generating new individuals representing new solutions, with the aim of finding optimal or approximate solutions to a problem [3].

Genetic programming starts with an initial population of potential solutions (good or bad), known as individuals. In each generation, these individuals are evaluated based on a fitness function that quantifies how well they solve the given problem. Individuals with higher fitness are more likely to be selected for reproduction, where crossover (recombination) and mutation operations occur, similar to the processes of genetic evolution [3].

The genetic programming approach allows the exploration of a broad solution space in search of effective solutions for complex and multidimensional problems. It is applied in various fields, including optimization, machine learning, and modeling.

C. Genetic Programming Applied to Decision Trees

Genetic programming (GP) applied to decision trees represents an innovative approach in the field of artificial intelligence. In this paradigm, decision trees are portrayed as chromosomes, enabling the evolution of effective solutions for multiclass classification problems. [7] emphasizes that this genetic representation facilitates the application of evolutionary operators, such as crossover and mutation, to generate new generations of decision trees, allowing the discovery of novel and improved solutions to the addressed problem.

The evolutionary process unfolds over iterations, where trees are selected for a reproduction pool, forming pairs that crossbreed to produce new individuals. Trees that are more adapted, as per a fitness function, have higher chances of being chosen for reproduction. This evolutionary approach aims to find decision trees that optimally fit the data patterns. Nguyen et al. [8] underscore the importance of a well-defined fitness function to efficiently guide the evolutionary process.

The advantages of this approach include the ability to handle complex problems and the flexibility to evolve decision tree structures without the need for manual definition. However, challenges such as the potential uncontrolled growth of the tree (overfitting) need to be addressed. [9] discuss strategies, such as penalties in the fitness function, to mitigate these challenges and ensure more generalized solutions.

In summary, genetic programming applied to decision trees offers a promising approach to solve multiclass classification problems, combining the flexibility of genetic evolution with the structured representation of decision trees. However, the careful selection of parameters and strategies to prevent overfitting is crucial in the development and implementation of this technique.

III. Methodology

A. Used Database

The Tennessee Eastman Process Simulation database is widely recognized as a benchmark in the field of process engineering and fault detection. Developed by the Oak Ridge National Laboratory in the United States, this database was designed to allow the evaluation and comparison of fault detection, diagnosis, and prediction algorithms and methods in a simulated environment of a complex chemical process [10]. Researchers employ this dataset to test and compare anomaly detection algorithms, pattern identification, and diagnosis in a chemical process scenario, fostering advancements in the field [11].

In total, the original database has 55 columns, with 54 input attributes and 1 output attribute. The column that presents the output attribute is called "faultNumber", representing the fault number, ranging from 0 to 21. This expresses a classification of 22 fault classes, where class 0 means no fault, and the other classes (1 to 21) represent the fault classification number.

In the present study, a database derived from a fault detection model based on Qualitative Trend Analysis (QTA), proposed by [12], was used. This model adopts a two-step process for fault detection in the Tennessee Eastman database. The first step uses the fuzzy set theory, while the second one relies on a Bayesian approach for detecting change points in time series, providing an indication of a possible fault. If this indication is established, the proposed model takes responsibility for identifying the specific nature of the fault.

Additionally, 22 variables were eliminated from the dataset in two phases by [12]. In the first phase, a correlation matrix obtained from the 52 input variables in the author's used database was employed. Variables with a correlation below 0.6 were eliminated, resulting in a reduction of 15 variables.

In the second phase, 7 more variables that showed no indications of faults by the Fuzzy/Bayesian approach were discarded. In other words, in the calculation of the new probability vector for change points for these variables, no changes were detected. Consequently, the vectors were zeroed out, resulting in the variables having only zero values, which does not affect fault classification.

Thus, 30 input variables were retained, which were used to train and test the classifier proposed in this work.

In the end, the dataset proposed by [12] presented 4180 instances, with 30 input attributes and 1 output attribute. In this sampling, all classes have 200 instances, except for classes 1, 9, 15, 19, 18, and 20, which have 199, 190, 198, 195, 199, and 199 instances, respectively.

B. Developed Algorithm

Using the concepts of decision trees and genetic programming, a predictive model was developed using Python. Three classes were created in total: 'Node', which stores the nodes of the tree, 'DecisionTree', that represents the classification trees, and finally, the 'PG' class (Genetic Programming). This contains the genetic operators that create the tree population and evolve them with the aim of finding better solutions for the fault classification problem.

When executed, the algorithm takes training parameters and the maximum number of iterations as inputs. It can also receive additional parameters such as the maximum depth of the trees, population size, crossover rate, mutation rate, elitism, the number of individuals participating in tournaments during the selection phase, and the number of split points for individuals during the crossover phase.

The pseudocode of the developed algorithm can be seen in Fig. 1.

```
Algorithm 1 Genetic Programming(TrainingData, MaxIt)
1:  $Pop_1 \leftarrow GenerateInitialPopulation(TrainingData)$;
2:  $g \leftarrow 1$;
3:  while $g < MaxIt$ do
4:      $Pop_g \leftarrow CalculateFitness(Pop_g)$;
5:      $Selected \leftarrow Selection(Pop_g)$;
6:      $NewGeneration \leftarrow Crossover(Selected)$;
7:      $NewGeneration \leftarrow Mutation(NewGeneration)$;
8:      $NewGeneration \leftarrow Elitism(NewGeneration)$;
9:      $Pop_{g+1} \leftarrow NewGeneration$;
10:     $g \leftarrow g + 1$;
11: end while
12: Return $Pop_g$
```

Fig. 1. Developed algorithm

In line 1 of the pseudocode, we have the first function to be called, which is *GenerateInitialPop()* that takes the training data as a parameter, aiming to generate the initial population randomly.

After the initial population is formed, the algorithm enters a loop, which lasts for the specified number of generations. In each generation within the loop, the population goes through the *Evaluation()* function (line 4), which calculates the fitness level of each individual. Right after, the *Selection()* (line 5) occurs, to select individuals for the *Crossover()* phase (line 6) through tournaments, creating a new generation, where individuals may undergo the *Mutation()* process (line 7). After the new generation is formed, the *Elitism()* function (line 8) is called, with the goal of saving the best individual from the previous generation and placing it in the new generation. Finally, the current population is replaced by the new generation (line 9), and the current generation number is incremented (line 9).

At the end of the loop, the final population found by the algorithm is returned (line 12), containing the last generation of individuals found by the model. From this, it is possible to select the best individual or individuals from this population to perform tests using test data, evaluating the test accuracy of the tree found, representing how well the tree performed in predicting fault classifications.

C. Generation of The Initial Population

The generation of the initial population is done through the *GenerateInitialPop()* function, which takes the training data as a parameter. In this function, a number of decision tree individuals equivalent to the user-specified number are created.

The construction of a tree is based on the training input data (instances/input attributes) and output (fault class). From this, nodes are randomly generated, where the input attribute related to this node, the split threshold, and the data split operator are randomly chosen. The function also checks various stopping conditions, such as the maximum tree depth, the minimum number of samples for a split, and whether all samples belong to a single class. If the stopping condition is met, the next node to be generated will be a leaf node, representing a class to be predicted.

Fig. 2 represents a decision tree generated by the function. In this example, it can be observed that the root node has the attribute 24 (equivalent to column 24 of the database) randomly selected, where the threshold was randomly chosen as 0.03, and the operator was <. When analyzing the set of training instances, if the value of column 24 of the instance is < 0.03, the instance proceeds to the left node; otherwise, it proceeds to the node on the right, and this process repeats until the instance reaches a leaf node, where the predicted class will be determined.

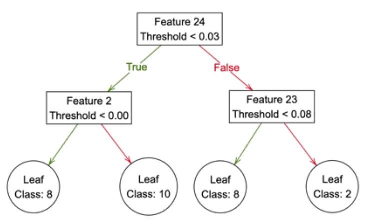


Fig. 2. Example of generated decision tree.

D. Fitness Function

To calculate the fitness of each decision tree in the population, the Misclassification Error criterion was used to measure the accuracy of the individual. This accuracy serves as its fitness. For this purpose, the *CalculateFitness()* function utilizes the training data, where the tree uses the input data to predict the outputs (fault classes). After the prediction, the predicted outputs are compared with the true outputs of the training data. In other words, it quantifies the ratio of correctly classified instances compared to the total instances in the dataset, providing an overall measure of the model accuracy in predicting the correct classess. This accuracy value is the fitness of the individual. This calculation is performed for each individual in the population.

E. Selection and Crossover

The selection operator chosen is tournament selection. In this method, there will be a number of tournaments equivalent to the size of the population. For each tournament, two individuals from the current population are randomly selected and compete against each other. The one with the higher fitness wins the tournament, and a copy of it is added to a list of winners. An individual may be drawn for competition more than once. Moreover, the number of competing individuals per tournament can be changed by the user.

After the tournaments and the list of winners are complete, the individuals undergo crossover. In this phase, two random individuals are taken from the list of winners to undergo crossover. If the crossover probability is equal to or greater than the defined crossover rate, crossover occurs; otherwise, the function returns the two randomly selected individuals. If crossover occurs, the two individuals are fragmented at one or more random points, and these segments are exchanged between the individuals, generating two new offspring individuals, which are then returned by the function. The number of crossovers that occur is equivalent to half the population size. Each return from the crossover function (two resulting individuals) is added to the list of the new generation. Consequently, a new generation is formed.

An example of a cross between two individuals can be seen in Fig. 3. In this example, in Individual 1, the node 'Feature 9' was segmented from the individual. Meanwhile, in Individual 2, it was the 'Feature 2' node that underwent segmentation. Following the segmentation process, the individuals crossbreed, giving rise to two new children, thus exchanging segments between them. Consequently, 'Child 1' is a copy of 'Individual 1,' but it now includes the 'Feature 2' node where 'Feature 9' used to be. On the other hand, 'Child 2' is a copy of 'Individual 2,' but with the 'Feature 9' node now in the place of 'Feature 2.' These two new individuals represent fresh solutions to the problem.

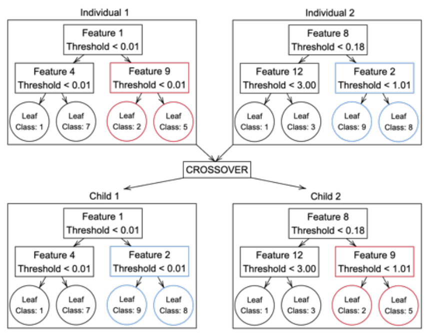

Individual 1
Feature 1 Threshold < 0.01
Feature 4 Threshold < 0.01
Feature 9 Threshold < 0.01
Leaf Class: 1
Leaf Class: 7
Leaf Class: 2
Leaf Class: 5
Individual 2
Feature 8 Threshold < 0.18
Feature 12 Threshold < 3.00
Feature 2 Threshold < 1.01
Leaf Class: 1
Leaf Class: 3
Leaf Class: 9
Leaf Class: 8
CROSSOVER
Child 1
Feature 1 Threshold < 0.01
Feature 4 Threshold < 0.01
Feature 2 Threshold < 0.01
Leaf Class: 1
Leaf Class: 7
Leaf Class: 9
Leaf Class: 8
Child 2
Feature 8 Threshold < 0.18
Feature 12 Threshold < 3.00
Feature 9 Threshold < 1.01
Leaf Class: 1
Leaf Class: 3
Leaf Class: 2
Leaf Class: 5

Fig. 3. Example of crossover between trees.

F. Mutation

In this phase, each individual in the new generation has the possibility of undergoing mutation. Observing the mutation rate defined by the user, if a randomly generated decimal number is greater than or equal to this rate, the individual undergoes mutation. In this process, a node of the tree is randomly selected, and its attribute, threshold, and split operator are randomly modified. If the selected node is a leaf node, the target class of that node is randomly modified among the possibilities, which range from 1 to 21.

Fig. 4 illustrates an example of an individual that was selected for mutation. In this individual, the node 'Feature 9' was chosen and underwent changes. Previously, this node used input attribute 9, with a threshold of 0. After the mutation, this node now uses input attribute 7, with a threshold of 2.31, thus becoming the 'Feature 7' node.

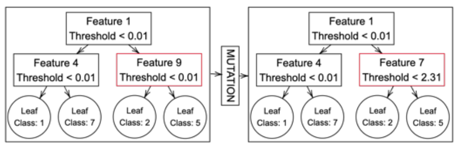

Feature 1 Threshold < 0.01
Feature 4 Threshold < 0.01
Feature 9 Threshold < 0.01
Leaf Class: 1
Leaf Class: 7
Leaf Class: 2
Leaf Class: 5
MUTATION
Feature 1 Threshold < 0.01
Feature 4 Threshold < 0.01
Feature 7 Threshold < 2.31
Leaf Class: 1
Leaf Class: 7
Leaf Class: 2
Leaf Class: 5

Fig. 4. Tree mutation example.

G. Elitism

After the new generation is formed, to prevent the loss of the best individual from the previous population, the elitism technique is applied. The individual with the lowest fitness in the new generation is replaced by the individual with the highest fitness from the current generation. This prevents the population from degrading rapidly in quality.

IV. TESTS AND RESULTS

After the algorithm implementation, tests were initiated. To achieve this, the database was divided using a stratified sampling strategy into training and testing sets, allocating 70% of the data for training and 30% for testing purposes. Additionally, the following parameters were defined:

- Maximum tree depth: 100.
- Number of generations: 1,000 generations.
- Population size: 200 individuals.
- Crossover rate: 0.9.
- Mutation rate: 0.6.

- Elitism enabled.
- Tournament selection (2 competing individuals).
- Single-point crossover.

With the above settings, 100 algorithm executions were performed, and the average of the obtained results was calculated. The results were:

- Average run time: 02:54:41.
- Average number of features used: 29.4.
- Medium depth: 41.4.
- Average number of nodes: 157.
- Average training accuracy: 0.772.
- Average test accuracy: 0.755.

The Fig. 5 presents the average test accuracies for each class, with a standard deviation of 0.2682 in this case. It can be observed that the prediction for classes 3 and 9 had an average accuracy below 0.2, with class 9 being the worst predicted by the model. Additionally, classes 20 and 21 had averages lower than 0.6, and classes 4, 11, 13, 16, and 18 achieved an average accuracy below 0.8. On the other hand, the remaining classes (12 in total) achieved accuracies higher than 0.8.

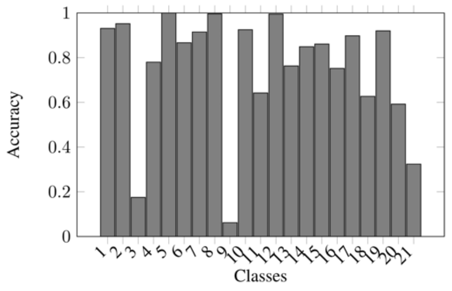


Fig. 5. Average test accuracy by class

Among the 10 tests conducted, the one with the best performance showed a training accuracy of 0.804 and a test accuracy of 0.799. Analyzing the Fig. 5 and the average of the results obtained, it is possible to notice that the average training accuracy obtained was 0.772, and the average test accuracy was 0.755, indicating these results as moderate. Regarding the average accuracy obtained per class, there were occurrences of extremely low accuracies, especially for classes 3 and 9, including accuracies equal to 0 in some of their executions, meaning that the resulting genetic algorithm made errors in all predicted classifications. On the other hand, 12 out of the 21 classes achieved accuracies higher than 0.8, indicating promising results.

V. Conclusions

This work aimed to present an approach for fault detection and classification, evaluating its performance when applied to the Tennessee Eastman Process. Decision trees induced by genetic programming were used to build and train the predictive classification model. The results of this application were collected and analyzed.

It is important to highlight that approaches based on decision trees can provide interpretable models, and the application of such models in the Tennessee Eastman Process has not been found in the previous literature. In this sense, this work stands as one of the first to use interpretable approaches in fault classification for the Tennessee Eastman Process dataset.

Another point to be discussed concerns the reduction of input attributes. By default, the dataset has 30 such attributes, and in a few executions, the proposed model managed to reduce this quantity to a maximum of 28 attributes. Although there was a reduction in some tests, this number is not significant or consistent.

Despite decision trees being simple models to understand and interpret, as their decisions are represented in a hierarchical structure that is easily comprehensible, facilitating explanations to non-technical users, the interpretability of the trees obtained by the model was hindered by their size. The trees had an average depth of 41.4 and an average number of nodes of 157. In light of these results, it identifies a greater difficulty in interpreting the resulting trees due to their size. Such size is also due to the complexity of the 21-class fault classification problem, which is an extensive issue.

Through the aforementioned ideas, it is concluded that, despite the model not achieving satisfactory results for all classes, a good part of the classes was predicted reasonably or adequately. Moreover, it is the first study that uses an interpretable model applied to the Tennessee Eastman dataset. However, the model needs changes and refinements for better results.

For future work, it is necessary to apply optimization techniques to improve the algorithm performance, aiming to reduce its execution time. Additionally, implementing functionalities and strategies that make the trees more interpretable and provide better accuracy results is crucial. The intention is to apply niche techniques, specifically fitness sharing based on Hamming distance, to increase the population diversity, and implement pruning techniques to reduce the size of the trees and make them more interpretable.

References

[1] A. Rajkomar, J. Dean, e I. Kohane, “Machine Learning in Medicine,” New England Journal of Medicine, vol. 380, no. 14, pp. 1347-1358, Abr. 2019.

[2] E. F. Brown et al., “Hierarchical decision trees for anomaly detection in interconnected systems,” in Proceedings of the International Conference on Industrial Engineering, pp. 126–132, 2020.

[3] J. R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, 1992.

[4] R. K. DeLisle and S. L. Dixon, “Induction of decision trees via evolutionary programming,” Journal of Chemical Information and Computer Sciences, vol. 44, no. 3, pp. 862–870, 2004.

[5] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, Classification and Regression Trees. Chapman & Hall, 1984.

[6] A. Silva, T. Killian, I. D. Jimenez Rodriguez, S. Son, e M. Gombolay, “Optimization Methods for Interpretable Differentiable Decision Trees in Reinforcement Learning,” arXiv, 2019.

[7] J. R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, 1985.

[8] Q. U. Nguyen, M. Zhang, K. Zhang, and S. Li, “Evolutionary construction of decision trees for multiclass classification,” IEEE Transactions on Evolutionary Computation, vol. 19, no. 6, pp. 822–834, 2015.

[9] N. Javed, F. Gobet, e P. Lane, “Simplification of genetic programs: a literature survey,” *Data Mining and Knowledge Discovery*, vol. 36, no. 4, pp. 1279-1300, Abr. 2022.

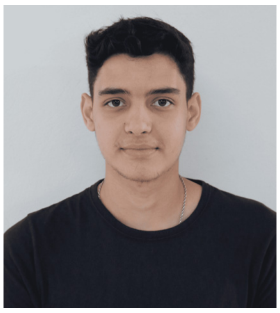
[10] R. W. J. Westerhout, F. J. J. Verhagen, and P. M. J. van den Hof, "Monitoring and diagnosis of industrial processes using chemomet- ric techniques," Computers & Chemical Engineering, vol. 27, no. 9, pp. 1259–1273, 2003.

[11] P. Wang and H. Wang, "A review of data-driven approaches for process systems fault detection and diagnosis," Computers & Chemical Engi- neering, vol. 94, pp. 188–200, 2016.

[12] M. F. D'Angelo, R. M. Palhares, R. H. Takahashi, and R. H. Loschi, "Fuzzy/bayesian change point detection approach to incipient fault detection," Control Theory & Applications, IET, vol. 5, pp. 539–551, 2011

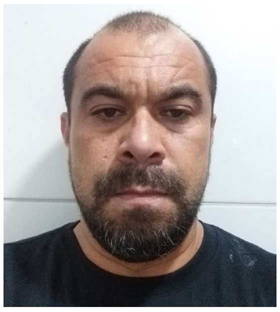
AUTHORS

Rogério Costa Negro Rocha



Rogério Costa Negro Rocha graduated with a Bachelor's degree in Information Systems in 2021 from the Federal Institute of Northern Minas Gerais, Salinas Campus. Currently, he is pursuing a Stricto Sensu Graduate Program in Computational Modeling and Systems at the State University of Montes Claros – UNIMONTES, Brazil, with an expected completion date by November 2024. He has prior experience in software development, web development, and mobile development, particularly focusing on systems tailored for enterprise and industrial management, as well as retail management and e-commerce projects. Additionally, he has worked on projects in the field of photometry, following agile Scrum methodologies. Moreover, he has academic experience in machine learning, natural computing, and evolutionary computation. His interests encompass software and mobile development, utilizing Java and Dart; web development with PHP, HTML, CSS, and JavaScript; evolutionary computing, leveraging Python, particularly employing genetic algorithms and genetic programming for resource allocation and optimization problems; and pattern recognition in images using Convolutional Neural Networks, also utilizing Python.

Laércio Ives Santos



Laércio Ives Santos graduated in Information Systems in 2008, obtained a Master's degree in Computational Modeling and Systems, and a Doctorate in Health Sciences in 2021, all from the State University of Montes Claros - UNIMONTES. Currently, he is a professor at the Federal Institute of Education, Science, and Technology of Northern Minas Gerais, in Montes Claros, mainly teaching in the courses of Computer Science, Information Technology, and Electrical Engineering. His main disciplines include: Database, Computer Programming, Artificial Intelligence, and Natural Computing. He has been a Visiting Professor in the Computational Modeling Program at UNIMONTES since 2022. His experience includes research using Machine Learning and Artificial Intelligence techniques for diagnosing engine and process failures, as well as predicting events related to the medical field. His areas of interest include: Evolutionary Computing, Swarm Intelligence, Artificial Neural Networks, ensemble learning, as well as relational and NoSQL databases, and software development in languages such as Python, MATLAB, and JavaScript.

AUTHORS

Rafael Almeida Soares

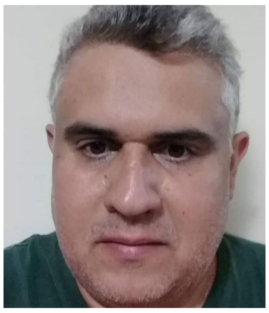
Rafael Almeida Soares completed his Bachelor's degree in Information Systems in 2022 at the Federal Institute of Education, Science, and Technology of Northern Minas Gerais, Salinas Campus. Currently, he is enrolled in the Stricto Sensu Graduate Program in Computational Modeling and Systems at the State University of Montes Claros, pursuing a Master's degree, expected to be completed by December 2024. Professionally, Rafael holds experience in backend development and mobile app development. He collaborated with the doctoral program in Health Sciences at the State University of Montes Claros to develop an application for childhood vaccination. His professional interests lie in technology applied to agriculture, livestock farming, and healthcare. Rafael's academic and professional pursuits extend to pattern recognition in images using Convolutional Neural Networks. He has already developed a system for license plate recognition. Moreover, he is keenly interested in machine learning, optimization, evolutionary computing, and the development of web and mobile applications.

Franciele Alves Barbosa

Franciele Alves Barbosa graduated with a Bachelor's degree in Information Systems in 2021 from the Federal Institute of Northern Minas Gerais, Salinas Campus. Currently, she is pursuing a Stricto Sensu Graduate Program in Computational Modeling and Systems at the State University of Montes Claros – UNIMONTES, Brazil, with an expected completion date by December 2024. She has prior experience in data mining, machine learning, time series forecasting, software development, web development, and mobile development. She has worked on a research project involving clustering reference evapotranspiration time series for the state of Minas Gerais using the K-means and Ward algorithms. Currently, she has worked on a project that utilizes artificial intelligence for healthcare, with a focus on cervical cancer diagnosis. Moreover, she has academic experience in machine learning, time series forecasting, deep learning and data science. Her interests encompass data mining, deep learning, time series forecasting and in the python programming language.

AUTHORS

Marcos F. Silveira V. D'Angelo



Marcos Flávio Silveira Vasconcelos D'Angelo joined the State University of Montes Claros in 2000 as an Associate Professor in Information Science. He earned his B.S. and M.S. degrees in Electrical Engineering from the Pontifical Catholic University of Minas Gerais in 1998 and 2000, respectively, and his Ph.D. in Electrical Engineering from the Federal University of Minas Gerais in 2010. His main research interests encompass dynamic systems, optimization theory and applications, and soft computing. Specifically, his work has focused on maintenance engineering. D'Angelo served as the coordinator of the systems engineering course and was a member of the university council at the State University of Montes Claros. Currently, he serves as a reviewer for journals and conferences, both nationally and internationally. To date, he has published approximately 49 full papers in journals, 4 chapters in books, and 45 full papers in conference proceedings. Additionally, he has coordinated research projects funded by grants and technological innovation projects.