Interpretable Components Using Genetic Programming Employing Instruction-like Structure

Arthur Hiratsuka Rezende[1], **Thiago Ambiel**[1], **Rafael Souza e Silva**[1], **André C. P. L. F. de Carvalho**[1]

[1]Institute of Mathematics and Computer Sciences – University of São Paulo (USP)
São Carlos – SP – Brazil

{arthurhr, thiago.ambiel, rafaelsoz}@usp.br, andre@icmc.usp.br

Abstract. *This paper introduces a novel feature extraction method, IGP, that generates components through both linear and non-linear combinations of features using Genetic Programming (GP). Unlike traditional GP approaches that rely on expression trees, IGP utilizes an instruction line structure. The study evaluates IGP's performance against 5 established feature extraction methods across 23 datasets, encompassing binary and multiclass classification tasks. The results demonstrate that IGP excels in several instances, particularly in binary classification, with further analysis exploring how the relationship between the number of classes, features, and instances contributes to its performance. Additionally, the scope for future investigations of IGP are commented.*

1. Introduction

The need for auditable machine learning applications, in compliance with legislation in Brazil[1] and the European Union[2], for example, demands the search for solutions that can be easily interpretable. The feature extraction in well-established algorithms, such as encoders and PCA, occurs through the combination of all the variables involved. This can diminish the interpretability [Gambella et al. 2021, Zhang et al. 2021] of the generated components or the interpretation of the variables involved in the decision-making process.

Advances in telecommunications, such as 5G, 6G, and IoT devices, combined with the increased use of sensors and data capture tools [Shafique et al. 2020], have resulted in high-dimensional databases concerning both features and instances. Dimensionality reduction can facilitate the use of simpler algorithms that require lower computational capacity, while aiming to maintain or improve performance.

A widely used approach for feature extraction is genetic programming, which typically employs operation trees to combine the original features, with the leaves representing the original features and the nodes representing mathematical operations. This relatively simple approach allows for linear or non-linear combinations of features.

This study proposes a novel methodology for feature extraction using genetic programming, to the best of the authors' knowledge. The innovation lies in the use of an instruction-based structure, derived from compiler processing applications, rather than the widely used operation tree approach, and it does not require crossover operations.

[1]Regulation on the use of AI - Bill No. 2338/2023.
[2]*AI Act* for regulation on the use of AI from 03/2024.

The tree structure is widely employed for solving problems hierarchically and has been successfully applied in numerous areas, including search algorithms, data sorting, and complex decision-making models like Random Forest. However, this approach imposes certain restrictions, particularly in how information is accessed and propagated, as each node usually access information from the nodes in the immediately preceding layer.

We formulated the hypothesis that this limitations can be overcome by implementing instruction-based modeling (IGP), similar to a programming language, where every node can access all available information in the model and in the data. It is believed that this will facilitate operations between components, compared to the use of trees.

The main objective of this work is to verify the hypothesis that the IGP can achieve results as good as those obtained with five baseline methods (PCA, t-SNE, autoencoder, MDS and LLE) for feature extraction. Additionally, it is expected that the proposed method will offer the advantage of easier interpretability of the generated components, maintaining performance in a classification task. If the proposed method proves successful, future research will explore whether it has advantages over the use of operation trees.

2. Background and Related Work

Evolutionary computation can be applied in several scenarios [Espejo et al. 2010], and numerous strategies can optimize this process [Zhan et al. 2022], such as reducing problem complexity through single-objective fitness functions or accelerating convergence time by utilizing pre-calculated solutions (implemented through internal memory in agents), strategies adopted in the present study, among others described in this section.

2.1. Functions, Complexity and Optimization

The selection of potential functions to generate components and the determination of components quantity are crucial [Kishore et al. 2000, Muni et al. 2004]. These studies indicate that using simple arithmetic functions (+, -, *, /) is preferable over non-linear approaches, which may have been limiting during the studies period.

The use of simple functions is not universally agreed upon among researchers and depends largely on the study's primary objectives. In cases where a significant reduction in solution complexity is desired, often in optimization-heavy scenarios (e.g., 500 epochs) [Ma and Teng 2019, Ma and Gao 2020, Ma et al. 2023, Meng et al. 2024], only the aforementioned simple functions are utilized.

However, instances using logical/conditional functions (max, min) and non-linear functions (sin(x), ReLU, etc.) [Lensen et al. 2019, Lensen et al. 2020, Wang et al. 2014] have successfully created latent spaces with clear class separation, effectively exploring search spaces. In this context, the present study aims to explore this condition, hypothesizing that clearer separation between class examples in the new space will help identify variables that are crucial for describing the data.

Regarding the complexity of generated functions, the phenomenon of bloat [Ma and Gao 2020, Meng et al. 2024] in GP is known, where machine learning model performance does not improve with overly complex functions, leading to loss of interpretability of constructed features. Controlling the complexity of feature-calculating functions can be achieved through the genetic algorithm's fitness function [Meng et al. 2024],

minimizing the number of operations. Empirical evidence suggests that lower complexity enhances performance and significantly reduces execution time.

2.2. Fitness Function - Filter and Wrapper Approaches

There are three possible paradigms for defining fitness functions in genetic programming for feature extraction. Filter methodologies utilize information theory measures, correlation, and distance, among others. Wrapper approaches employ a simple predictor model during component generation and use performance metrics (accuracy, number of correct predictions, etc.) as fitness functions. Lastly, hybrid solutions combine weighted sums of filter and wrapper measures, albeit with an additional hyperparameter to optimize.

Commonly used filter-based fitness functions include information theory measures [Wang et al. 2014, Ma and Teng 2019], correlation or distance measures [Meng et al. 2024]. Wrapper approaches often use accuracy rates [Ma and Teng 2019, Ma and Gao 2020], balanced accuracy [Zhang and Smart 2006], or the number of predictors correctly classifying [Kishore et al. 2000, Muni et al. 2004].

Some authors also consider complexity measures, such as the proportion of selected features from the total [Ma and Gao 2020], or limiting the number of operations in expression trees [Meng et al. 2024]. Other approaches use similarity fitness [Lensen et al. 2019, Lensen et al. 2020], aiming to maintain neighbor relationships in a high-dimensional space in a lower-dimensional similarity space. [Zhang and Smart 2006] uses measures of distance and areas of the distribution of the features for each class as a loss function, minimizing distribution overlap.

While studies like [Ma et al. 2023, Meng et al. 2024] suggest multi-criteria fitness in GP yields superior results compared to single-criteria fitness, hybrid strategies often assign greater weight to wrapper-derived components, typically 0.8 or higher, underscoring their importance. In comparative studies of wrapper, filter, and hybrid methods [Ma and Teng 2019], comparable performance is noted, with hybrids often favoring wrapper contributions, similar to findings in [Ma et al. 2023].

Given the study's goal to test the proposed methodology's feasibility using an instruction-like structure instead of expression trees to perform feature extraction, a simple wrapper strategy will be employed. The algorithm chosen is K-Nearest Neighbors (KNN) with 5 neighbors, using macro F1 score as the fitness metric to balance precision and recall.

3. Instruction-based Genetic Programming

3.1. Modeling

The data structure known as Expression Tree was developed to represent complex mathematical operations in programming language interpreters [Mitchell 1991]. Later, this structure was adapted to serve as a model base for exploring the sample space of arithmetic functions through genetic algorithms, giving rise to Genetic Programming Trees.

Similarly, we propose adapting the modeling used in the Instruction Selection paradigm [Cooper and Torczon 2023], employed by compilers to translate code into machine instructions, for representing complex arithmetic operations. This enables an alternative way to explore the sample space of arithmetic functions. In our method, we define three crucial domains for the model optimization process, as shown below.

1. Attribute Domain (α): This domain encompasses all the attributes that can be selected for an operation. With each new operation executed, a new attribute is added to this domain, representing the result of the executed operation.
2. Unary Operations Domain (θ): This domain includes the operations that can be performed using only one variable. In our experiments, we used the following operations: a, $|a|$, $sin(a)$, $cos(a)$, $log(a)$, e^a.
3. Binary Operations Domain (π): This domain includes operations that can be performed using two variables as input. In our experiments, we used the following operations: $a + b$, $a - b$, $a \times b$, a/b. It is important to note that the division operations are protected, meaning that if $b = 0$, the function returns 0.

With these domains defined, we obtain a discrete search space composed of the combination of possible values from these three main domains. This space can be utilized to find nonlinear relationships between input variables and the target variable.

For example, considering the function 1 that uses the variables $sepal_{length}$, $petal_{length}$, and $sepal_{width}$ from the classic Iris dataset.

$$\frac{sepal_{length}}{petal_{length} \times sepal_{width}} \tag{1}$$

We can model this function using only two simple instructions:

Table 1. Example of matrix modeling for a nonlinear expression with three variables from the Iris dataset and two division operations.

Line	**Node** (a)	**Operation Domain** $(\theta\pi)$		**Node Aux.** (b)
		π	θ	
1	$sepal_{length}$	$\frac{a}{b}$	NULL	$petal_{length}$
2	$\frac{sepal_{length}}{petal_{length}}$	$\frac{a}{b}$	NULL	$sepal_{width}$

The idea is to model simple operations that comprise a complex equation as a sequence of instructions, organized in a two-dimensional matrix.

1. The matrix columns represent the metadata of each instruction, including the variables involved and the operation applied to them.
2. The matrix rows represent the instructions, which are executed sequentially from the first to the last.

The return value from line 2 is the result calculated by the last executed instruction. This process is similar to how computers execute software, following the instructions of the code stored in memory. To improve the interpretability of this modeling, we can track the behavior of the output variable as a function of the number of executed instructions.

$$sepal_{length} \longrightarrow \frac{sepal_{length}}{petal_{length}} \longrightarrow \frac{sepal_{length}}{petal_{length} \times sepal_{width}} \tag{2}$$

3.2. Optimization

A simplified implementation of genetic programming (GP) is employed, operating solely with mutations and eliminating the need for crossover operations. This approach is based on the bio-inspired strategy of asexual reproduction [Farasat et al. 2010], which aims to reduce execution time. This strategy has also demonstrated interesting results when applied to genetic programming [Khanteymoori et al. 2021].

Furthermore, we apply a selection technique that favors solutions with fewer instructions to achieve a satisfactory outcome. The performance of the solutions is assessed, and if the absolute difference between their results is smaller than a threshold ϵ — set to 10^{-4} — the solutions are considered equivalent, and the one with the fewest instructions is selected as the best. The mutation process occurs by choosing a random operation line from an individual, and one of the following mutation operations is executed:

Table 2. Proposed mutation operations for our modeling.

Mutation	Domain
Select a new attribute to perform a unary or binary operation.	α
Replace a unary operation with a binary operation.	π
Replace a binary operation with a unary operation.	θ
Select a new auxiliary attribute to perform a binary operation.	α
Add a new random operation line to the solution.	α, θ, π

Since the search space is very complex and small changes can result in significant topological changes in the generated space, we opted not to use crossover operations between solutions. Mutation operations alone are sufficient to make significant structural changes. The goal of defining this search space for complex mathematical expressions is to find interpretable nonlinear transformations that project the data into a low-dimensional space where the classes of the target attribute are well separated, as shown in Figure 1.

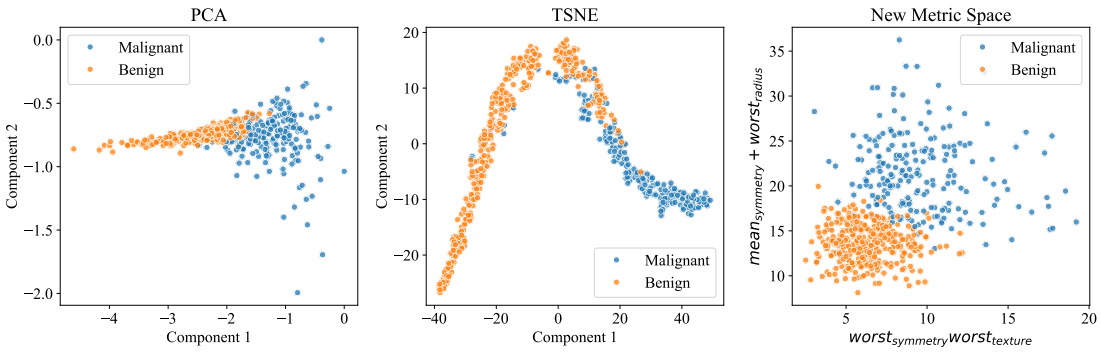

Figure 1. Example of low-dimensional spaces generated by the PCA, t-SNE methods, and our algorithm IGP on the Breast Cancer dataset. The better the separation of points of different colors, the easier it is to solve the problem.

By finding this type of transformation, we can extract valuable information from the data, such as creating new attributes for simpler classification models and identifying

the most influential features on the target attribute, even if this influence relationship is not linear. To evaluate these transformations, we use the mean F1-Score in a 5-fold cross-validation, employing the KNN model with 5 neighbors. This approach allows robust evaluation of the quality of the transformations, ensuring that the new attributes truly enhance the model's ability to capture nonlinear interactions between the variables.

4. Experiments

The conducted experiment aims to compare the proposed method with five well-established dimensionality reduction algorithms to verify if the proposed method has satisfactory performance. To evaluate if there are statistically significant differences in the performance of the algorithms, the non-parametric Wilcoxon signed-rank test is used for pairwise comparisons between the algorithms. The results are derived from a k-fold cross-validation and the training data is Z-score normalized.

4.1. Experimental Setup - Comparison with Established Methods

Due to the stochastic nature of the proposed method (IGP), three random initialization seeds are used. Evaluation metrics are obtained through 5-fold stratified cross-validation, with 70% of the data for training and 30% for testing. Two components are generated in all cases, which are used in conjunction with Logistic Regression for classification. The maximum number of iterations is defined as 200 and the newton-cg solver is adopted.

The 23 datasets selected for benchmarking, with their characteristics listed in Table 3, are collected from the UCI machine learning repository. The experiments use datasets with numerical and categorical features, ranging from 3 to 60 features. Additionally, the datasets contain between 150 and 6497 instances, with 8 of them for binary classification and 15 for multiclass classification (between 3 and 10 classes).

Table 3. Dataset characteristics and metadata, Class/Feat denotes the number of classes divided by the number of features and Class/Inst denotes the number of classes divided by the number of instances

ID	Dataset	Classes	Features	Instances	Class/Feat	Class/Inst
1	Bank Note	2	4	1372	0.500	0.001
2	Blood Transfusion	2	4	748	0.500	0.003
3	Mammographic Mass	2	5	961	0.400	0.002
4	Rice	2	7	3810	0.286	0.001
5	AIDS Clinical Trials	2	23	2139	0.087	0.001
6	Breast Cancer	2	30	569	0.067	0.004
7	Ionosphere	2	34	351	0.059	0.006
8	Mines vs Rocks	2	60	208	0.033	0.010
9	Iris	3	4	150	0.750	0.020
10	Vertebral Column	3	6	310	0.500	0.010
11	Wholesale Customers	3	7	440	0.429	0.007
12	Website Phishing	3	9	1353	0.333	0.002
13	Wine	3	13	178	0.231	0.017

Continued on next page

Table 3. Dataset characteristics and metadata (continued).

ID	Dataset	Classes	Features	Instances	Class/Feat	Class/Inst
14	Healthy Aging	3	14	714	0.214	0.004
15	Waveform	3	21	5000	0.143	0.001
16	Land Mines	5	3	338	1.667	0.015
17	User Knowledge	5	5	403	1.000	0.012
18	Glass	6	9	214	0.667	0.028
19	Dermatology	6	34	366	0.176	0.016
20	Wine Quality	7	11	6497	0.636	0.001
21	Ecoli	8	7	336	1.143	0.024
22	Yeast	10	8	1484	1.250	0.007
23	Cardiotocography	10	21	2126	0.476	0.005

The possible operations are those described in section 3.1, with a maximum of 3 instructions to avoid the bloat phenomenon described in section 2.1 (increased complexity of generated components translates to higher computational cost without performance gain). A population of 75 individuals, 8 epochs, and a tournament size of 10 is initialized. For the fitness wrapper function, which considers the F1 score obtained using KNN, the number of neighbors is set to 5. The mutations described in section 3.2 occur in all individuals (except the best one) in each epoch.

Below are listed the feature extraction and dimensionality reduction algorithms adopted for comparison and their respective hyperparameters, in the cases which differ from the default hyperparameters in the scikit-learn implementaiton [Pedregosa et al. 2011].

- Principal Component Analysis (PCA) [Jolliffe 2011]: Computes some linearly uncorrelated components, such that each successive component represents the axis of most remaining variance.
- T-distributed Stochastic Neighbor Embedding (t-SNE) [Van der Maaten and Hinton 2008]: Seeks to minimize the Kullback-Leibler divergence between the joint probabilities (obtained through converting the similarity between instances) of the low-dimensional embedding and the high-dimensional data. The perplexity value was set to 25.
- Multi-dimensional scaling (MDS) [Ingwer Borg 2005]: An algorithm that seeks to maintain the distances between instances observed in the higher-dimensional space in the newly transformed space. The Euclidean distance is used as dissimilarity measure.
- Locally Linear Embedding (LLE) [Roweis and Saul 2000]: An algorithm that models each instance in the high-dimensional space as a linear combination of its neighbors and seeks to maintain this combination in the reduced space. The solver used to compute the eigenvectors was the dense solver.
- Auto-encoder (AE) [Kramer 1991]: A neural network architecture that can be used for feature extraction and dimensionality reduction. A single-layer auto-encoder with a two-dimensional latent space, trained for 200 epochs with the Adam optimizer and a learning rate value of 1.0×10^{-3}.

4.2. Experimental Results

The accuracy and AUC results obtained are listed in Tables 4 and 5, respectively. Since only the IGP, PCA, and AE methods performed best on at least one dataset, the results presented are only for these methods. The complete experiment results, including t-SNE, MDS, and LLE, are available in a public repository.

Regarding the accuracy results in Table 4, the IGP performs best on 9 out of 23 datasets, is equivalent on 7, and inferior on 7, with PCA better in 4 cases and AE being superior in 3. In the binary classification datasets (1 to 8), IGP's results have higher accuracy in 4 cases, is equivalent in 3, and is outperformed by PCA in 1 case. For the multiclass classification datasets (9 to 23), there are 4 cases with no statistical difference between methods, with IGP being superior in 5, followed by PCA in 3 and AE in 3.

Table 4. Comparison of Accuracy values between IGP, PCA, and AutoEncoder. p-PCA and p-AE denote the p-values for PCA and AutoEncoder, respectively, regarding Wilcoxon's test with the proposed IGP method.

Dataset	IGP	PCA	AE	p-PCA	p-AE	Result
1	**0.982 ± 0.000**	0.758 ± 0.000	0.749 ± 0.001	0.000	0.000	+ +
2	0.767 ± 0.000	**0.772 ± 0.000**	0.771 ± 0.000	0.030	0.028	- -
3	*0.811 ± 0.000*	*0.806 ± 0.001*	*0.810 ± 0.000*	0.378	0.712	= =
4	*0.928 ± 0.000*	0.926 ± 0.000	0.923 ± 0.000	0.155	0.011	= +
5	**0.807 ± 0.000**	0.761 ± 0.001	0.780 ± 0.000	0.000	0.008	+ +
6	*0.940 ± 0.000*	*0.952 ± 0.001*	*0.944 ± 0.000*	0.125	0.570	= =
7	**0.831 ± 0.001**	0.585 ± 0.006	0.586 ± 0.001	0.000	0.000	+ +
8	**0.671 ± 0.003**	0.590 ± 0.006	0.587 ± 0.006	0.003	0.003	+ +
9	**0.938 ± 0.001**	0.916 ± 0.002	0.911 ± 0.003	0.026	0.035	+ +
10	**0.812 ± 0.002**	0.680 ± 0.009	0.672 ± 0.002	0.003	0.003	+ +
11	*0.717 ± 0.000*	*0.718 ± 0.000*	*0.718 ± 0.000*	0.373	0.373	= =
12	0.653 ± 0.000	**0.799 ± 0.009**	0.783 ± 0.000	0.000	0.001	- -
13	0.923 ± 0.000	0.959 ± 0.002	**0.959 ± 0.001**	0.033	0.041	- -
14	*0.521 ± 0.001*	*0.525 ± 0.000*	*0.524 ± 0.001*	0.649	0.733	= =
15	0.735 ± 0.000	0.866 ± 0.001	**0.866 ± 0.000**	0.000	0.000	- -
16	**0.447 ± 0.002**	0.284 ± 0.005	0.282 ± 0.001	0.000	0.000	+ +
17	**0.880 ± 0.001**	0.528 ± 0.002	0.542 ± 0.002	0.000	0.000	+ +
18	0.466 ± 0.003	0.584 ± 0.008	**0.589 ± 0.002**	0.000	0.000	- -
19	0.685 ± 0.000	**0.800 ± 0.014**	0.770 ± 0.003	0.004	0.044	- -
20	**0.501 ± 0.000**	0.460 ± 0.000	0.461 ± 0.000	0.000	0.000	+ +
21	0.756 ± 0.002	**0.792 ± 0.004**	0.787 ± 0.001	0.018	0.029	- -
22	*0.446 ± 0.001*	*0.445 ± 0.002*	*0.451 ± 0.000*	0.454	1.000	= =
23	*0.527 ± 0.000*	*0.518 ± 0.004*	0.515 ± 0.000	0.733	0.762	= =

Analyzing the AUC results in Table 5, the IGP stands out in 9 datasets, has equivalent performance in 7, and worse performance in 7, PCA being superior in 5 cases and AE in 2. In binary classification datasets, the IGP has superior results in 4 cases, and there is no statistical difference in the other 4. In the multiclass datasets, the IGP performs better in 5 cases, is equivalent in 3, and is outperformed by PCA in 5 and by AE in 2.

Table 5. Comparison of AUC values between IGP, PCA, and AutoEncoder. p-PCA and p-AE denote the p-values for PCA and AutoEncoder, respectively, regarding Wilcoxon's test with the proposed IGP method.

Dataset	IGP	PCA	AE	p-PCA	p-AE	Result
1	**0.980 ± 0.000**	0.715 ± 0.001	0.703 ± 0.003	0.000	0.000	+ +
2	*0.126 ± 0.011*	*0.177 ± 0.006*	*0.162 ± 0.007*	0.050	0.268	= =
3	*0.788 ± 0.001*	*0.799 ± 0.000*	*0.803 ± 0.001*	0.303	0.188	= =
4	*0.938 ± 0.000*	*0.935 ± 0.000*	0.933 ± 0.000	0.151	0.012	= +
5	**0.477 ± 0.045**	0.135 ± 0.003	0.346 ± 0.006	0.000	0.041	+ +
6	*0.917 ± 0.002*	*0.935 ± 0.001*	*0.922 ± 0.001*	0.169	0.670	= =
7	**0.878 ± 0.003**	0.717 ± 0.001	0.718 ± 0.001	0.000	0.000	+ +
8	**0.630 ± 0.017**	0.556 ± 0.004	0.550 ± 0.005	0.021	0.021	+ +
9	**0.938 ± 0.002**	0.915 ± 0.002	0.910 ± 0.003	0.016	0.027	+ +
10	**0.754 ± 0.012**	0.524 ± 0.002	0.545 ± 0.002	0.000	0.000	+ +
11	*0.278 ± 0.000*	*0.279 ± 0.000*	*0.279 ± 0.000*	0.373	0.373	= =
12	0.430 ± 0.009	**0.551 ± 0.000**	0.543 ± 0.000	0.000	0.000	- -
13	0.925 ± 0.002	0.960 ± 0.000	**0.961 ± 0.001**	0.022	0.026	- -
14	0.248 ± 0.001	**0.279 ± 0.001**	0.278 ± 0.001	0.001	0.002	- -
15	0.735 ± 0.001	0.866 ± 0.000	**0.866 ± 0.000**	0.000	0.000	- -
16	**0.403 ± 0.006**	0.211 ± 0.001	0.213 ± 0.001	0.000	0.000	+ +
17	**0.722 ± 0.006**	0.393 ± 0.003	0.411 ± 0.002	0.000	0.000	+ +
18	0.263 ± 0.006	**0.346 ± 0.001**	0.343 ± 0.001	0.001	0.002	- -
19	0.596 ± 0.017	**0.768 ± 0.002**	0.709 ± 0.005	0.000	0.026	- -
20	**0.179 ± 0.000**	0.147 ± 0.000	0.147 ± 0.000	0.000	0.000	+ +
21	*0.506 ± 0.009*	*0.522 ± 0.005*	*0.485 ± 0.003*	0.599	0.421	= =
22	*0.311 ± 0.005*	*0.289 ± 0.001*	*0.329 ± 0.001*	0.252	0.489	= =
23	0.346 ± 0.007	**0.431 ± 0.000**	0.414 ± 0.000	0.001	0.005	- -

These results indicate that IGP performs better especially in binary classification, with superior results in 4 out of 8 cases and no statistical difference in at least 3 others. For multiclass cases, IGP shows superior/equivalent results in 9 out of 15 cases (accuracy) and in 8 out of 15 cases (AUC). These findings suggest that IGP generates components that lead to superior/equivalent performance compared to well-established methods. Besides, the components involve a reduced number of features (there are only 3 instruction lines), making IGP versatile and effective for binary and multiclass datasets.

To explore the types of data for which IGP performs best, an analysis is conducted considering the class ratio concerning the number of features and instances. Considering the class/instance ratio, as shown in Figure 2, it is possible to observe that in the range of 0.005 to 0.015, the IGP is superior in 5 out of 7 cases, both in terms of accuracy and AUC, and in the remaining 2 cases, there is no statistical difference with the other methods.

For datasets that fall above the 0.015 threshold, IGP is better in only 1 out of 5 (multiclass) cases, the dataset 9-iris (150 instances, 4 features, and 3 classes) both in terms of accuracy and AUC. The cases that IGP has worse performance are on 13-wine (178x13x3), 18-glass (214x9x6), 19-dermatology (366x34x6), and 21-ecoli (336x7x8).

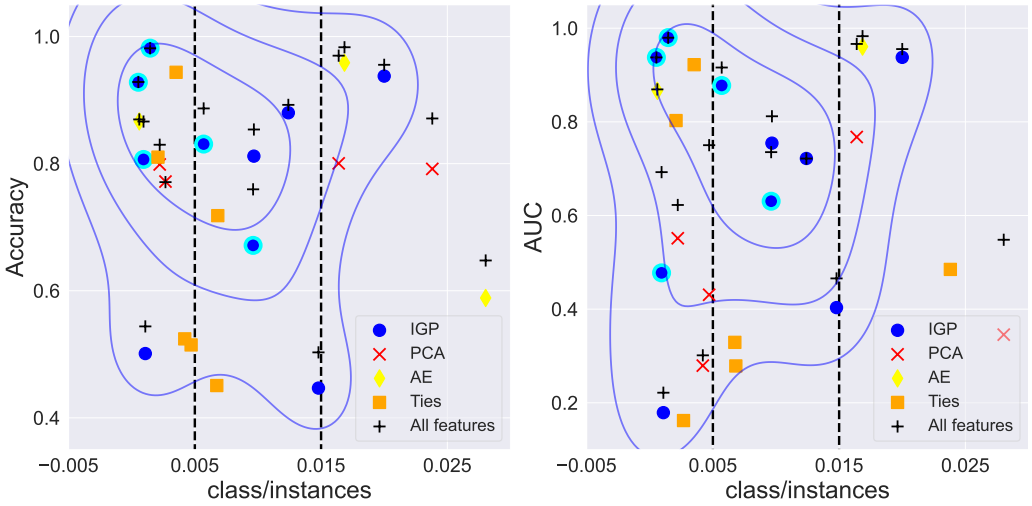

Figure 2. Meta-analysis regarding the proportion between the number of classes and instances, the highlighted IGP refers to binary classification

Analyzing the relationship between the number of classes and the number of features, as shown in Figure 3, the proposed IGP method performed better in 6 out of 10 cases when considering the threshold above 0.5. Additionally, in this region only in the datasets 18-glass (214x9x6) and 21-ecoli (336x7x8) the IGP are not the best method, with ties in the cases 2-blood (748x4x2) and 22-yeast (1484x8x10).

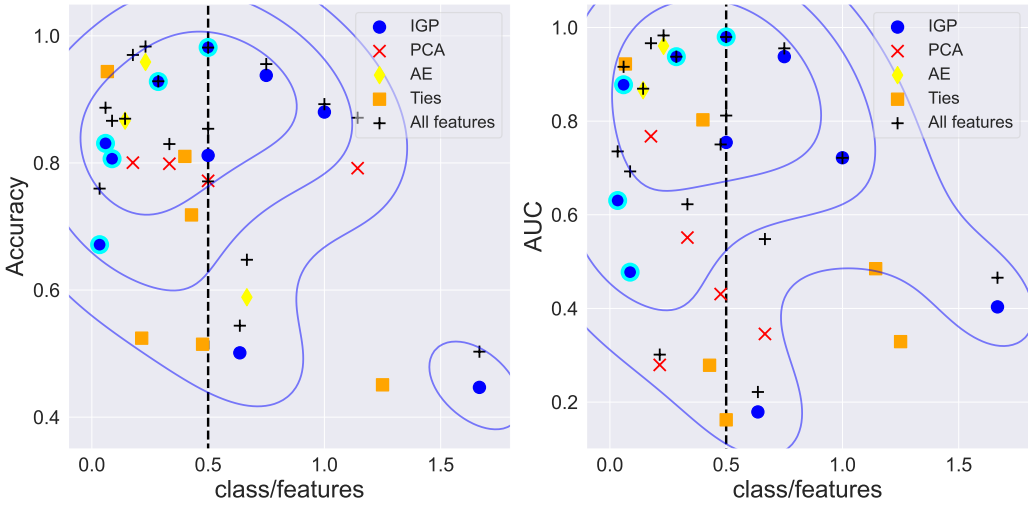

Figure 3. Meta-analysis regarding the proportion between the number of classes and features, the highlighted IGP refers to binary classification

It is noted that datasets 18 (glass) and 21 (ecoli) are multiclass and are among the six cases with the fewest instances, in addition to having similar numbers of classes and features. In other cases, no clear pattern is observed, suggesting that other factors may influence the quality of the generated components. The quantity and methodology of categorical feature treatment, the statistical distribution of the data, and other factors will be analyzed in future investigations.

5. Discussion and Conclusion

A method for dimensionality reduction through the creation of components obtained via linear and non-linear combinations of the original features using Genetic Programming is proposed. The novelty of the IGP method lies in using an instruction line structure instead of expression trees, which are commonly implemented in GP.

This work aims to evaluate whether the proposed method performs comparably to established methodologies in the literature for feature extraction. A comparison is made with 5 different methods, using 23 diverse datasets for classification tasks, both binary and multiclass. The datasets contain both categorical and numerical features.

The results indicate that our algorithm performs better in some cases, particularly for binary classification datasets, when the class-to-instance ratio falls within intermediate values (between 0.005 and 0.015) and when the class-to-feature ratio exceeds the 0.50 threshold. For binary cases, IGP shows superior/equivalent results in 7 out of 8 cases, and in multiclass cases, IGP shows superior/equivalent results in 9 out of 15 cases (accuracy) and in 8 out of 15 cases (AUC).

Given that the proposed method demonstrates good performance for feature extraction tasks, the next steps involve optimizing the hyperparameters of the IGP, testing different classification algorithms, and exploring variations in the fitness function, which are already underway. After this stage, it will be possible to compare our method with established GP frameworks that use expression trees to confirm the hypothesis that the proposed modeling approach may offer advantages over the usual GP modeling.

References

Cooper, K. D. and Torczon, L. (2023). Chapter 11 - instruction selection. In Cooper, K. D. and Torczon, L., editors, *Engineering a Compiler (Third Edition)*, pages 575–616. Morgan Kaufmann, Philadelphia, third edition edition.

Espejo, P. G., Ventura, S., and Herrera, F. (2010). A survey on the application of genetic programming to classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(2):121–144.

Farasat, A., Menhaj, M. B., Mansouri, T., and Moghadam, M. R. S. (2010). Aro: A new model-free optimization algorithm inspired from asexual reproduction. *Applied Soft Computing*, 10(4):1284–1292. Optimisation Methods Applications in Decision-Making Processes.

Gambella, C., Ghaddar, B., and Naoum-Sawaya, J. (2021). Optimization problems for machine learning. *European Journal of Operational Research*, 290(3):807–828.

Ingwer Borg, P. J. F. G. (2005). *The Four Purposes of Multidimensional Scaling*. Springer New York, New York, NY.

Jolliffe, I. (2011). Principal component analysis in international encyclopedia of statistical science. *Berlin, Heidelberg: Springer Berlin Heidelberg*, pages 1094–1096.

Khanteymoori, A., Alamdar, F., and Ghorbani, F. (2021). Arp: asexual reproduction programming. *Connection Science*, 33(2):256–277.

Kishore, J. et al. (2000). Application of genetic programming for multicategory pattern classification. *IEEE Transactions on Evolutionary Computation*, 4(3):242–258.

Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243.

Lensen, A. et al. (2019). Can genetic programming do manifold learning too? In Sekanina, L., Hu, T., Lourenço, N., Richter, H., and García-Sánchez, P., editors, *Genetic Programming*, pages 114–130, Cham. Springer International Publishing.

Lensen, A., Zhang, M., and Xue, B. (2020). Multi-objective genetic programming for manifold learning: balancing quality and dimensionality. *Genetic Programming and Evolvable Machines*, 21(3):399–431.

Ma, J. and Gao, X. (2020). Designing genetic programming classifiers with feature selection and feature construction. *Applied Soft Computing*, 97:106826.

Ma, J., Gao, X., and Li, Y. (2023). Multi-generation multi-criteria feature construction using genetic programming. *Swarm and Evolutionary Computation*, 78:101285.

Ma, J. and Teng, G. (2019). A hybrid multiple feature construction approach for classification using genetic programming. *Applied Soft Computing*, 80:687–699.

Meng, W. et al. (2024). Ensemble classifiers using multi-objective genetic programming for unbalanced data. *Applied Soft Computing*, 158:111554.

Mitchell, R. J. ("1991"). *"Expression Trees"*, pages "219–231". "Macmillan Education UK", "London".

Muni, D., Pal, N., and Das, J. (2004). A novel approach to design classifiers using genetic programming. *IEEE Transactions on Evolutionary Computation*, 8(2):183–196.

Pedregosa, F. et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326.

Shafique, K., Khawaja, B. A., Sabir, F., Qazi, S., and Mustaqim, M. (2020). Internet of things (iot) for next-generation smart systems: A review of current challenges, future trends and prospects for emerging 5g-iot scenarios. *IEEE Access*, 8:23022–23040.

Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(11).

Wang, P. et al. (2014). Multiobjective genetic programming for maximizing roc performance. *Neurocomputing*, 125:102–118. Advances in Neural Network Research and Applications Advances in Bio-Inspired Computing: Techniques and Applications.

Zhan, Z.-H. et al. (2022). A survey on evolutionary computation for complex continuous optimization. *Artificial Intelligence Review*, 55(1):59–110.

Zhang, M. and Smart, W. (2006). Using gaussian distribution to construct fitness functions in genetic programming for multiclass object classification. *Pattern Recognition Letters*, 27(11):1266–1274. Evolutionary Computer Vision and Image Understanding.

Zhang, Y., Tiňo, P., Leonardis, A., and Tang, K. (2021). A survey on neural network interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 5(5):726–742.