SN Computer Science (2024) 5:91
https://doi.org/10.1007/s42979-023-02106-3

ORIGINAL RESEARCH

Exploring SLUG: Feature Selection Using Genetic Algorithms and Genetic Programming

Nuno M. Rodrigues[1] · João E. Batista[1] · William La Cava[2] · Leonardo Vanneschi[3] · Sara Silva[1]

Received: 7 October 2022 / Accepted: 29 June 2023

Abstract
We present SLUG, a recent method that uses genetic algorithms as a wrapper for genetic programming and performs feature selection while inducing models. SLUG was shown to be successful on different types of classification tasks, achieving state-of-the-art results on the synthetic datasets produced by GAMETES, a tool for embedding epistatic gene–gene interactions into noisy datasets. SLUG has also been studied and modified to demonstrate that its two elements, wrapper and learner, are the right combination that grants it success. We report these results and test SLUG on an additional six GAMETES datasets of increased difficulty, for a total of four regular and 16 epistatic datasets. Despite its slowness, SLUG achieves the best results and solves all but the most difficult classification tasks. We perform further explorations of its inner dynamics and discover how to improve the feature selection by enriching the communication between wrapper and learner, thus taking the first step toward a new and more powerful SLUG.

Keywords Feature selection · Epistasis · Genetic programming · Genetic algorithms · Wrapper · Learner · Machine learning

Introduction

Epistasis can generally be defined as the interaction between genes, and it is a topic of interest in molecular and quantitative genetics [1]. In machine learning (ML), several types of epistatic interactions have been studied. In evolutionary computation, epistasis has traditionally been interpreted as the interaction between characters, sets of characters or, generally speaking, parts of the chromosome representing solutions. This type of epistatic interaction has attracted the interest of researchers mainly because of its effect on fitness landscapes and, consequently, problem hardness. The topic has been studied since the early 90s (see, for instance, [2, 3]), and one of the most popular outcomes of those studies was the *NK*-landscapes benchmark [4], in which the amount of epistasis is tunable by means of two parameters, *N* and *K*. This benchmark has been used in several circumstances for testing the performance of genetic algorithm (GA) variants (see for instance [5–10], just to mention a few), and more recently, it has also been extended to genetic programming (GP) [11].

An in-depth, although not very recent, survey of studies of epistasis in GA can be found in [12]; while in [13], the effect of epistasis on the performance of GA is critically

This article is part of the topical collection "Evolution, the New AI Revolution" guest edited by Anikó Ekárt and Anna Isabel Esparcia-Alcázar.

✉ Nuno M. Rodrigues
nmrodrigues@fc.ul.pt

João E. Batista
jebatista@fc.ul.pt

William La Cava
william.lacava@childrens.harvard.edu

Leonardo Vanneschi
lvanneschi@novaims.unl.pt

Sara Silva
sara@fc.ul.pt

[1] LASIGE, Department of Informatics, Faculty of Sciences, University of Lisbon, Lisbon, Portugal

[2] Harvard Medical School, Boston Children's Hospital, Boston, MA, USA

[3] NOVA Information Management School (NOVA IMS), Universidade Nova de Lisboa, Campus de Campolide, 1070-312 Lisbon, Portugal

Published online: 12 December 2023

revised, highlighting the difficulty of GA in optimizing epistatic problems. In [14], epistasis was used to select the appropriate basis for basis change space transformations in GA, and in the same year [15] proposed a method to decipher the exact combinations of genes that trigger the epistatic effects, focusing on multi-effect and multi-way epistasis detection. Recently, a new benchmark was proposed [16] where epistasis-tunable test functions are constructed via linear combinations of simple basis functions. A different way of interpreting epistasis in ML is by studying the interactions between features in data. The problem of attribute interdependency is well known in ML. It has been studied in several approaches, using, for instance several types of correlation [17] or mutual information [18].

In this paper, we tackle a rather different type of problem: we want to be able to deal with datasets where, among many variables, only a very limited number of them are useful and able to explain the target, and they must necessarily and only be used together for the model to be accurate. In other words, all the few "important" variables must be selected, while the many "confounding" ones must be left out. If only one (or more) of the "important" variables are left out, then the model is not able to perform better than a simple random guest. Furthermore, these few important variables are not necessarily correlated between each other, or have any other relationship of interdependency. This type of behavior can be observed, for instance, in some of the Korn's benchmark problems proposed in [19], or in some medical problems, where finding epistasis can be crucial to identify the association between disease and genetic variants, and consequently be able to develop medical treatments and prevention [20]. It is a common intuition that, for problems characterized by such a typology of data, feature selection plays a crucial role. However, many of the existing feature selection algorithms will fail, for not being able to catch the epistatic relation between features. The objective of this work is to present a feature selection strategy that, integrated in a very natural way with the modeling algorithm, is appropriate for working with epistatic datasets.

The epistatic datasets studied in this paper have been generated using the GAMETES algorithm, introduced in [21], and have already been used in [22] as a benchmark to validate the M4GP classification method. Similar types of datasets have also been studied in [23], where a GP-based pipeline optimization tool (TPOT-MDR) was proposed to automatically design ML pipelines for bioinformatics problems. For tackling problems characterized by this type of data, Urbanowicz and colleagues recently presented RelieF-based feature selection [24], a unique family of filter-style feature selection algorithms that are sensitive to feature interactions and that can be applied to various types of problems, including classification and regression. In [22], this method has been coupled with M4GP, achieving state-of-the-art results on the tested GAMETES datasets.

Our proposal consists of using a GA for feature selection. The idea, presented for instance in [25–27], is framed in a well-established research track, and surveys can be found in [28, 29]. With the proliferation of data and the consequent development of ML, the use of GA for feature selection increased in the last decade. Numerous recent contributions can be found, for instance, aimed at improving the method in presence of vast amounts of data [30, 31], or applying the method in several different real-world scenarios, including medicine [32], economy [33], image processing [34], remote sensing [35] and sociology [36], just to mention a few. However, in this work, we match the GA with another evolutionary algorithm, Genetic Programming (GP), obtaining an integrated, and purely evolutionary, method that is able to perform feature selection and at the same time induce good models using the selected features. The GA part acts as a wrapper to the GP part, that is the learner. We call our approach SLUG (which stands for feature SeLection Using Genetic algorithms and genetic programming), and compare it to both standard GP and other GP-based algorithms already used on the GAMETES datasets, such as M3GP [37] and M4GP [22]; we also compare it with other GA-wrapped ML classifiers that also perform feature selection, such as decision trees, random forests, and XGBoost.

Although substantially different from SLUG, some related methodologies have been studied in the past. In [38], the opposite of SLUG was proposed, a methodology where GP was used for feature selection and GA for feature construction. In [39], GP is wrapped around GP itself with the objective of reducing the dimensionality of a dataset with one million features while also making use of GPU hardware to speed up the computation. In [40], the authors proposed a competitive and cooperative coevolutionary approach (Symbiotic Bid-Based GP) to perform both attribute subspace identification and classifier design at the same time. This approach was able to reduce the total attribute space from hundreds of thousands to no more than seventy features, while the classifiers themselves only used five to seven features. In [41], the authors proposed Cooperative Co-Evolutionary Genetic Programming, a new framework for high-dimensional problems. The proposed framework performs co-evolution at three different levels (genotype, feature, output) and the results showed that performance improves only when the output level cooperation happens.

Prior research had focused on the selection of features in high-dimensional datasets, and a notable example of a reliable benchmark is the feature selection challenge presented at the Neural Information Processing Systems (NEURIPS) conference in 2003 [42]. The aforementioned challenge evaluated the efficacy of various model combinations from multiple teams on five UCI datasets with different levels of dimensionality, ranging from 500 to 100,000 features. Besides establishing robust baselines on what models and feature combinations are better suited for this task, which provided interesting insights for future research, it was possible to conclude that sometimes eliminating extraneous features is not critical to achieving a good classification performance, with some of the solutions using all the available features.

Our proposal also consists of using GP for further feature selection and for classification. Similar to the work of several authors that use GP-based methods for classification in high-dimensional data, such as eGP [43], M3GP [44], M4GP [22], GP-SVI [35] and the standard GP itself [39, 45–47], SLUG [48] also showed promising results in this type of problem.

SLUG was first introduced in [48], where it has shown to be successful on different types of classification tasks, achieving state-of-the-art results on several GAMETES datasets. It has also demonstrated that the GA as wrapper and GP as learner are the right elements that grant SLUG its success. In this paper, we extend that initial work, by reporting the previous results and testing the original SLUG on six further GAMETES datasets of increased difficulty. Also, we study and discuss the inner dynamics of SLUG and describe a simple and promising improvement to its feature selection process, proposing an enriched combination of wrapper and learner for a future and more powerful SLUG. The following are the main contributions of the current work:

- We demonstrate the effectiveness of SLUG on six additional GAMETES datasets of increased difficulty, showing the applicability of the method on a wider range of epistatic problems;
- We conduct a comprehensive analysis of the inner workings of SLUG, also proposing an enriched combination of wrapper and learner that is expected to improve the performance of SLUG on high-dimensional datasets.

SLUG

The proposed method, feature **S**e**L**ection **U**sing **G**enetic algorithms and genetic programming (SLUG), uses a cooperative approach that joins these two evolutionary algorithms. Each GA individual is represented using a binary chromosome of the same length as the total number of features in the original dataset. This chromosome encodes a selection of features, where an allele equal to 1 means that the corresponding feature is used, while an allele equal to 0 means that the corresponding feature is not used. The quality of each GA individual is assessed by running GP with the features selected by that GA individual. More specifically, the fitness of the GA individual corresponds to the fitness of the best GP individual at the end of this GP run. A graphical representation of the SLUG pipeline is shown in Fig. 1, with the evaluation of the individuals being detailed in Fig. 2. As it is customary, once the GA individuals have been evaluated, a new GA population is formed by applying selection and the genetic operators, and after a number of generations the GA terminates and returns both the chromosome with the best selected features and the GP model that achieved the best results using only those features as input data (Fig. 1). Finally, the best GP model is evaluated on the test dataset using the features selected by the GA.

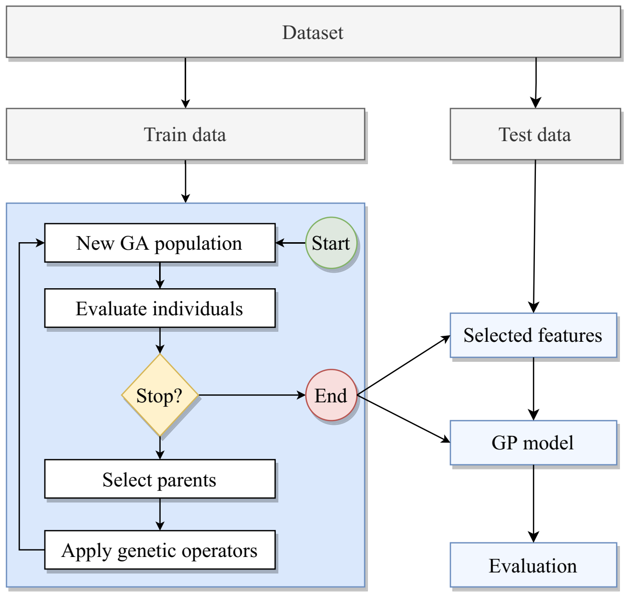


Fig. 1 A graphical representation of the SLUG pipeline

Naturally, the GP model does not have to use all the GA-selected features, since GP also performs its own feature selection during its evolution. In fact, this is one of the strengths of SLUG for epistatic datasets. The number of informative features on the GAMETES datasets is so low that not even a method like GP, which has feature selection abilities, can isolate them from the numerous other ones. So, in SLUG the GA only has the task of reducing the number of features that GP can potentially use, so its task is facilitated. In other words, the strength of SLUG is that the feature selection step performed by the GA does not need to be accurate: as long as the right features are among a reasonable number of selected ones, GP can do the rest of the job. We may think of the feature selection process of SLUG as composed of two phases: a sort of "preselection" (or explicit feature selection) performed by the GA in a preprocessing phase (where hopefully all the informative features are maintained, but many of the non-informative ones are eliminated), and the final feature selection (or implicit feature selection) performed by GP at learning time, whose

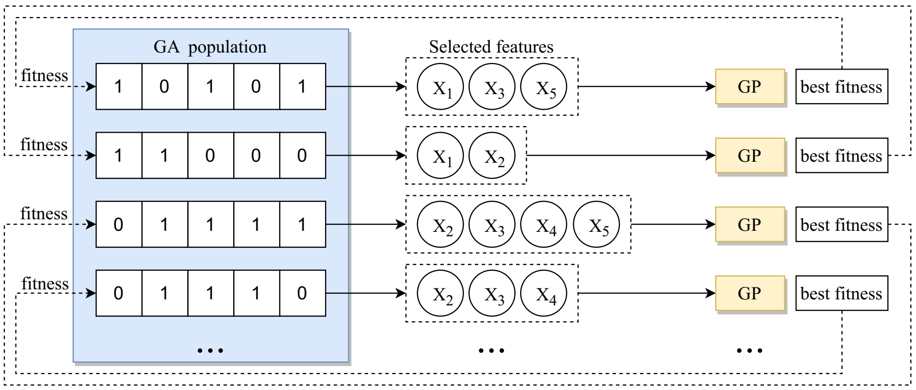


Fig. 2 Illustration of the way the GA individuals are evaluated by running GP with only the selected features. The best fitness of the GP run is the fitness of the respective GA individual

objective is to keep all and only the informative features in the final model.

The main limitation of SLUG, promptly identified in [48], is its high computational cost, caused by the obvious fact that an entire GP run is needed for evaluating each GA individual. Some alternatives are discussed in Sect. "Wrappers and Learners". Given that the goal is not, and never was, to obtain the best possible model, but rather to identify the epistatic features, one way to reduce the running time could be to accelerate the convergence and stop the evolution once they are found. Recognizing the moment in which the key features are found has prompted an exploration of the evolutionary dynamics of SLUG, described in Section "Accuracy and Key Features", which ultimately uncovered another limitation: once the key features are found by GP, they may be lost because the GA is not informed of which GP-selected features were responsible for the reported fitness. Section "Feature Selection Pressure" proposes a possible solution for this problem.

Data

We test the SLUG method on two distinct sets of problems: regular and epistatic. For the first set, we use four standard binary classification problems: HRT (Heart) [49]; ION (Ionosphere) [49], PRK (Parkinsons) [49] and SON (Sonar) [50]. Details regarding the composition of these datasets can be found in Table 1.

For the second set, we use GAMETES datasets, which are a collection of simulated gene–disease association datasets produced by a tool for embedding epistatic gene–gene interactions into noisy genetic datasets [51]. GAMETES generates random, pure, strict n-locus models, and respective simulated datasets for these models. According to [51], an n-locus model is purely and strictly epistatic if all n loci, but no fewer, are predictive of disease status.

We use 16 different problems that vary according to three measures of difficulty: number of epistatic loci (2, 3), which represent the key genes/features that solve the problem (the higher the number, the harder the problem); number of features (10, 100, 1000) with binary values 0 or 1 (the higher the number of features, the more difficult it is to find the few essential ones); signal-to-noise ratio (0.05, 0.1, 0.2, 0.4), which refers to the degree of separation between the true signal and the noise (the higher the signal-to-noise ratio, the easiest it is to locate the key epistatic features). Each problem consists of a perfectly balanced binary classification task where a two-way or three-way epistatic interaction (2 or 3 loci) is present but is masked by the presence of confounding features and noise, and the label predicts the presence or absence of the disease.

Due to computational and time constraints, we did not perform experiments on all the possible combinations of number of features and signal-to-noise ratio. We selected ten two-way (2w) datasets and six three-way (3w) datasets, and named them (joining the information on features and ratio): 2w_10_005, 2w_10_01, 2w_10_02 and 2w_10_04; 2w_100_005, 2w_100_01, 2w_100_02 and 2w_100_04; 2w_1000_02 and 2w_1000_04; 3w_10_01 and 3w_10_02; 3w_100_01 and 3w_100_02; 3w_1000_01 and 3w_1000_02.

The main advantage of the GAMETES data when compared with the NEURIPS data mentioned earlier [42] is that it offers a controlled experimental environment, allowing for the manipulation of the measures previously mentioned to assess the limits of each model. In contrast, the NEURIPS data comprise varied and uncorrelated datasets. Nonetheless, we believe that conducting additional research evaluating the models explored in this study on datasets like the

Table 1 Number of features, observations, and negative/positive ratio on each dataset

Datasets	HRT	ION	PRK	SON
Features	13	33	23	61
Observations	270	351	195	208
Neg/Pos Ratio	45/55	65/35	75/25	46/54

ones utilized in the NEURIPS challenge could offer valuable insights and further strengthen our claims.

Methods

Besides standard GA and standard GP, which are part of the SLUG method, we also compare our results with the following GP-based methods:

M3GP: M3GP stands for multidimensional multiclass GP with multidimensional populations [37]. Originally designed for multiclass classification, in M3GP each individual is composed of a mutable number of trees, also called dimensions, from which we extract a set of hyper-features that are then given to a classifier. Along with the standard crossover and mutation operators, M3GP includes an additional crossover, which swaps dimensions between individuals, and two additional mutations, which add/remove dimensions to/from an individual. The fitness of each individual is calculated by running a classifier on the hyper-feature space created by the trees of the individual. On the original implementation of M3GP, this is by default the Mahalanobis distance classifier.

M4GP: While the M3GP uses a tree-based structure for the individuals, M4GP, the successor of M3GP, uses a stack-based structure, which naturally provides support for multiple outputs. Regarding genetic operators, M4GP uses stack-based operators that are equivalent to the ones used by M3GP. For selection, M4GP uses lexicase selection, which outperformed standard tournament selection, and age-fitness Pareto survival selection in experiments [22].

M4GP+EKF: Expert knowledge filter (EFK) is a preprocessing feature selection algorithm from the RelieF family [52]. In M4GP+EKF it is used to reduce the dataset to the top ten features before giving it to the M4GP algorithm [22]. Since EKF is applied only as a preprocessing operation, it causes only some residual overhead and does not affect the training time. From now on, we will call this variant M4GP-E.

As part of the discussion, we also present some results obtained by replacing the GP part of SLUG with other ML methods, namely, decision trees (DT), random forests (RF), and extreme gradient boosting, better known as XGBoost (XGB). It should be noted that the DT, standard GP, M3GP and M4GP methods perform implicit feature selection by evolving models that do not use all available features. We can say the same about the RF and XGBoost models. However, due to the ensemble characteristic of the algorithms, more features are selected, mitigating their feature selection capabilities. As such, the principal feature selectors in this work are the GA part of the SLUG variants and the EFK part of the M4GP-E algorithm.

Experimental Setup

We run SLUG for 50 generations of the GA, using a population of 100 individuals. The GP populations also have 100 individuals, but they evolve for only 30 generations, which our initial experiments revealed to be sufficient to evaluate the quality of the selected features. GP uses the traditional binary arithmetic operators $[+, -, /, *]$ and no random constants. Fitness is the overall accuracy in the training set, measured after transforming the real-valued outputs of GP into class labels. The best fitness of each GP run is passed to the GA as the fitness of each individual, as explained in Sect. "SLUG", and therefore the GA (and therefore SLUG) also uses the overall accuracy as fitness (as do all the other GP and non-GP methods used here). Both GA and GP select the parents of the next generation using tournaments of size 5. Regarding the genetic operators, GP uses the standard subtree crossover and mutation with 50% probability each. GA also uses standard crossover that swaps same-sized blocks between 2 chromosomes with probability of 70%, and standard mutation that performs bit-flip on the chromosome with probability of $1/n$ (where n is the population size) and each bit has probability of $1/m$ of being flipped (where m is the length of the chromosome, i.e., the number of features of the problem). Both GA and GP use some elitism: GP guarantees that the best individual of one generation survives into the next; GA does not guarantee the survival of the best chromosome from one generation to the next, to avoid diversity loss, but it keeps track and returns the best chromosome (and respective GP model) that was ever achieved during the entire run.

Standard GP, M3GP, and both M4GP variants all use populations of 500 individuals evolving for 100 generations and, like SLUG, they all use tournaments of size 5. For more specific details on the M3GP and M4GP implementations and settings, the reader should consult Sect. "Methods" and the papers cited therein. The implementation of the GP methods will be available for download once the paper is accepted. The STGP, M3GP, M4GP, and SLUG implementations we use in this work can be found here.[1] Regarding the methods DT, RF, and XGB mentioned in the discussion, we use the implementations provided by Scikit-learn [53]. We perform hyperparameter optimization by means of grid search with fivefold cross-validation on the entire dataset, for each of the three methods. For DT, we optimize the split criterion and maximum depth; for RF, we optimize the split criterion, number of estimators, and maximum depth; for XGB, we optimize the learning rate, maximum depth, and number of

[1] https://github.com/jespb/Python-STGP, https://github.com/jespb/Python-M3GP, https://github.com/cavalab/m4gp-gametes and https://github.com/NMVRodrigues/SLUG.

Table 2 Median test overall accuracy of each method on the non-GAMETES binary classification tasks. Best results for each problem are identified in bold. Results with no statistically significant difference from the best are also highlighted in bold

	HRT	PRK	ION	SON
GP	0.778	0.831	**0.858**	0.698
M3GP	0.790	**0.881**	**0.873**	**0.786**
M4GP	0.784	**0.864**	**0.868**	**0.762**
M4GP-E	**0.802**	**0.873**	**0.854**	**0.738**
SLUG	**0.827**	**0.864**	**0.877**	0.730

estimators. The GA runs with the exact same parameters as SLUG. In all cases, we randomly split the datasets 30 times, one for each run, in 70% training and 30% test.

Results

We measure the overall accuracy of the methods and present the results as boxplots (training and test) of the 30 runs and tables with the (test) medians. To assess the statistical significance of the results, we perform one-way non-parametric ANOVA analysis by means of pairwise Kruskal–Wallis with Holm correction, using 0.05 as the significance threshold. The Appendix contains the Holm-corrected *p*-values obtained in all the comparisons.

Regular Classification Tasks

Taking into consideration the results presented in Table 2, Fig. 3, and Appendix Table 5, we can see that our approach performs well, on par with the other GP methods such as M3GP and M4GP. Compared to the baseline of standard GP, SLUG performs better on both HRT and PRK datasets, and presents no significant differences on the remaining two. Regarding the M3GP and M4GP baselines, the results are also positive, with SLUG outperforming both methods on one problem, presenting no significant difference on two others, and being outperformed in the remaining problem. Lastly, regarding M4GP-E, this method outperforms SLUG in one problem, and no significant difference was found between them in the remaining problems. Finally, we could not help but notice one thing that appears to be different between SLUG and most other methods, that is the consistently low dispersion of the results on training (observable in Fig. 3).

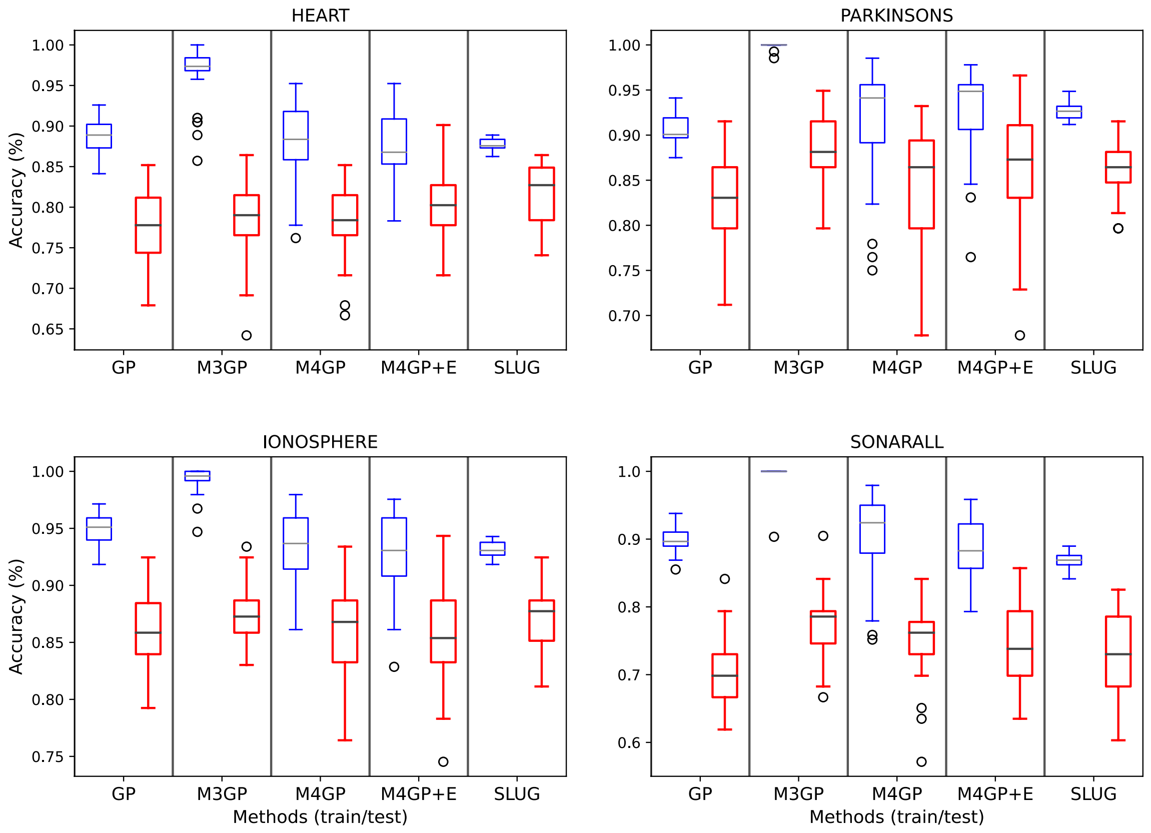


Fig. 3 Performance on the non-GAMETES binary classification datasets. Each plot contains, for each method, the results on the training (left) and test (right) sets

GAMETES Classification Tasks

Starting with the two-way epistatic datasets, taking into consideration the results presented in Table 3, Fig. 4 and Appendix Table 6, the first thing to notice is the fact that the standard GP baseline was one of the best methods on the 10-feature GAMETES problems. It outperformed both M4GP and M4GP-E on the 2w_10_005 dataset, M4GP-E on 2w_10_01, and all except SLUG on 2w_10_02.[2] We hypothesize that, on these easier problems, the exploration of different dimensional feature spaces that M3GP and M4GP perform is not helpful to the search, preventing the exploitation of better solutions.

Regarding our approach, the results were again highly positive, with SLUG invariably being one of the top-performing methods in all problems. The GA of SLUG is able to preselect a set of features which are then further filtered by the standard GP populations, also producing a ready-to-use model to apply to the problem.

On the 2w_1000_04 dataset, SLUG produced results significantly worse than M4GP-E. We attribute this to the default parameterization of SLUG, which always uses very small populations of 100 individuals. Particularly in the GA, this is too small to allow a proper exploration of the search space on the 1000-feature problems, making it harder for SLUG to filter out the redundant features. To confirm this hypothesis, we ran SLUG with a larger GA population of 200 individuals. Although this is the double of the previous population size, it is still a very low number of individuals for such a large search space (however, further increasing the size of the population becomes computationally demanding, an issue that is discussed later). We named this variation SLUG Large (SLUG-L). As seen on Fig. 5, particularly on the 2w_1000_04 dataset, SLUG-L is slightly improved, enough to be significantly better than the other solutions, and not significantly worse than M4GP-E. Once again we notice that SLUG exhibits a lower dispersion of results than most other methods (Fig. 4), this time not only on training but also on test.

Moving to the three-way epistatic datasets, the results presented in Table 4, Fig. 6 and Appendix Table 7 show that, once again, SLUG performs better than the other methods. On the 10-feature problems, it shares the best results with both M4GP variants (and with GP on 3w_10_01); on the 100-feature problems, it is the sole winner, clearly the only method performing better than random guess; on the 1000-feature problems, no method was significantly better than the others, all performing as bad as random guess.

[2] We performed 30 runs using the same total number of comparisons as SLUG using the standard GP (10000 individuals and 1500 generations). With this, the median test accuracy achieved was 0.4982, while the best was 0.5348.

Discussion

From the previous results, we can state that SLUG is a powerful method that performs feature selection while inducing high-quality models. On the set of four regular problems, it was one of the best methods in three of them. On the set of ten two-way GAMETES problems, it was always one of the best methods, although it required a larger population size on one of them. On this one exception, one of the hardest problems, the other winner besides SLUG-L was M4GP-E. Finally, on the six three-way GAMETES problems, SLUG was the only method that always ranked first, although failing to produce useful models for the 1000-feature problems, like all the other methods.

Wrappers and Learners

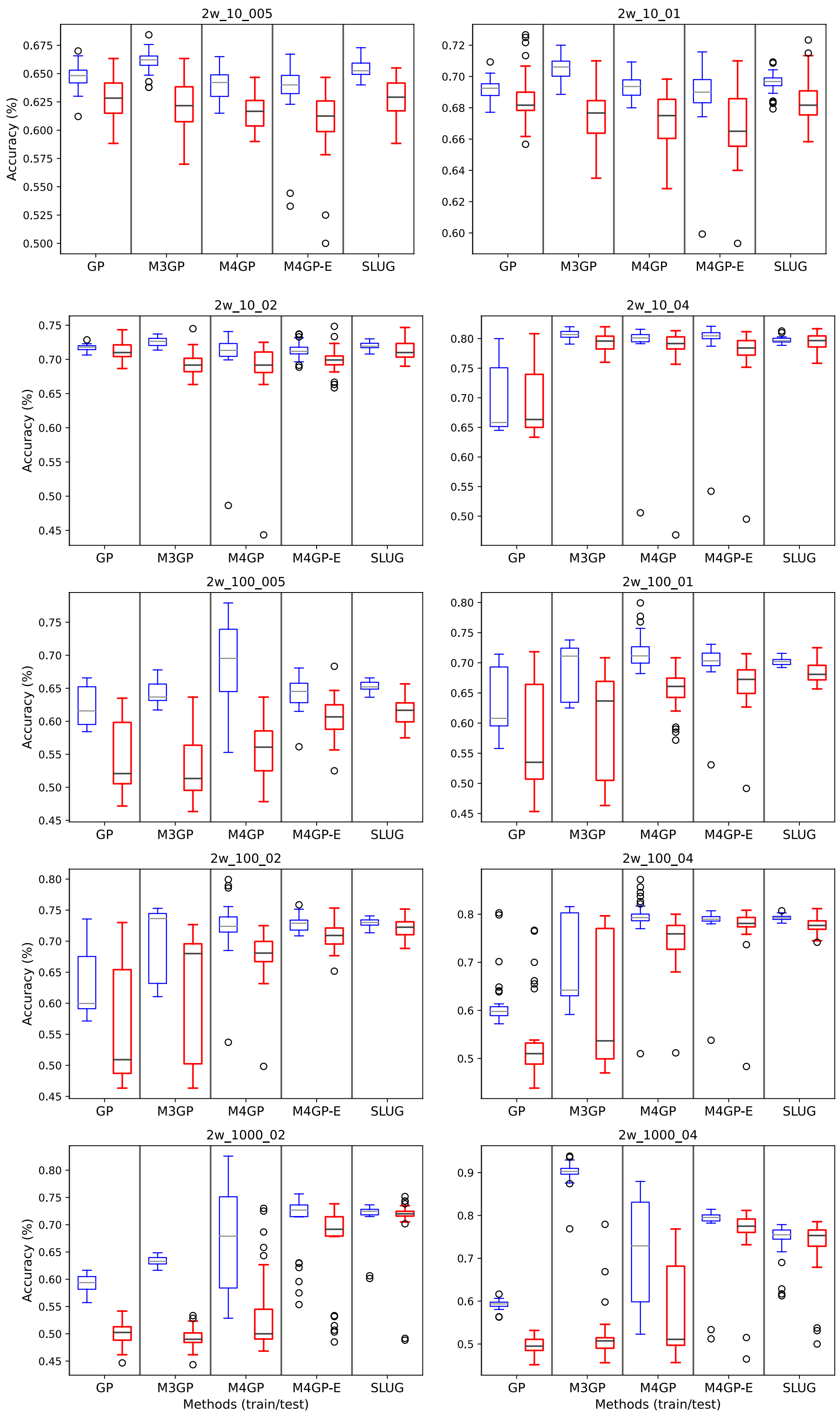
Published results on M4GP [22] had already shown that wrapping a feature selection method around a powerful classifier can improve the results significantly, and here we confirm that indeed, M4GP-E is often significantly better than M4GP. Reminding that SLUG is also the product of wrapping a feature selection method (GA) around a powerful classifier (GP), our results reconfirm the advantages of such an approach, since SLUG is very often significantly better than standalone GP.

Naturally, we are interested in searching for the best match between wrapper and learner, and we begin by exploring why SLUG performs so well; which of its parts is more important, the GA wrapper of the GP learner. On the one hand, we observe that M4GP is in general a stronger learner than GP; on the other hand, M4GP-E is not stronger than SLUG. Therefore, GA seems to be a better wrapper than the EKF used in M4GP-E, and mainly responsible for the success of SLUG.

While the combination of GA with M4GP seems like a promising match to explore in the future, for now we try to answer a simple question: is GA such a good wrapper that it can improve also the performance of other ML methods, arguably less powerful than the GP-based ones, like DT, RF, and XGB? This question is not only academically interesting, but also important from a practical point of view. Two evolutionary algorithms nested in each other is never an efficient solution in terms of computational effort, so it is not a surprise that SLUG is sluggish. Any of the three other mentioned ML methods runs much faster than GP, so wrapping GA around any of them could result in a much faster SLUG. Furthermore, like GP, these methods can also perform feature selection on their own, on top of the preselection made by GA.

Therefore, we experiment with alternative variants of SLUG where GP is replaced by DT, RF, and XGB. The problem chosen to test these variants is the GAMETES 2w_1000_04, coincidentally the one problem where SLUG-L

Fig. 4 Performance on the GAMETES two-way datasets. Each plot contains, for each method, the results on the training (left) and test (right) sets



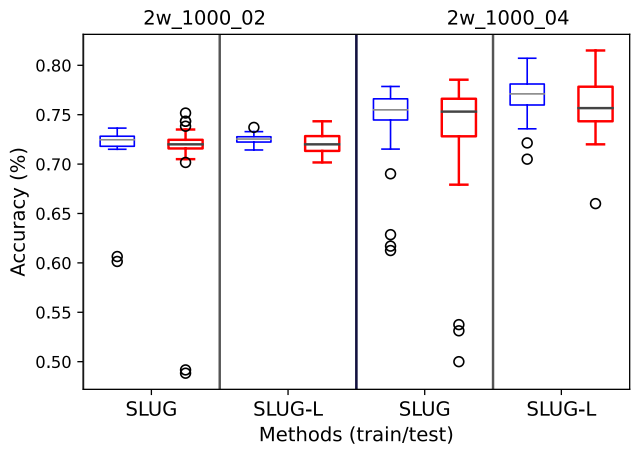


Fig. 5 Performance of different SLUG variants on the two higher dimensional GAMETES datasets. Each plot contains, for each variant, the results on the training (left) and test (right) sets

was required because SLUG was not one of the best methods (see Sect. "Results"). We chose this particular problem because it has already been used in previous studies [22, 43] where the standalone unwrapped versions of DT, RF, and XGB were unable to solve the problem.

The obtained results are shown in Fig. 7 and reveal that, even when wrapped with GA, these methods are not able to solve the problem, and this means that GP is also essential for the success of SLUG. Since the other methods also perform feature selection, the reason why GP is essential is not clear, particularly after observing that in one of the 30 runs the DT method, which is undoubtedly the less powerful one, was able to obtain a high-quality model (highest outlier in test, Fig. 7), and it did so after only 17 generations.

Accuracy and Key Features

Figure 8 shows the evolution of test accuracy (medians of the 30 runs) for all the 16 GAMETES problems tested here. The names of the problems are on the right, by the same order as the respective lines, and the colors, thickness, and markers are explained in the caption of the figure. Clearly, the difficulty of the problems is mostly driven by the number of epistatic loci (two-way problems mostly at the top and three-way at the bottom). On the two-way problems the difficulty is also driven by the signal-to-noise ratio; while for the three-way problems, it is driven by the number of features. As shown previously, SLUG achieves state-of-the-art results on most of these problems, except on the two 3w_1000 problems, the most difficult ones. The reason why SLUG fails on these problems is because, unlike on the other problems, it is rarely able to find the three key features. This is shown in Fig. 9, which plots the distribution of the number of key features included in the best GA individual at the end of the run, in the 30 runs. It can be observed that, whereas for all the two-way problems the two key features are always included (with two outliers on each of the 1000-feature problems), for the three-way problems this only happens on the 10-feature problems. On the 100-feature problems, the median is still the correct number of key features, but on the 1000-feature problems, the GA seldom includes all of them. Given this failure, we now perform further explorations of how the accuracy evolves with generations, and in the next section we discuss why the pressure for feature selection is lower than initially thought, and how it could be increased in order to improve SLUG.

Figure 10 shows the evolution of accuracy on two 1000-feature problems, one where SLUG succeeded (2w_1000_04) and one where it failed (3w_1000_02). The lines represent the medians of the 30 runs, while the boxplot-like representations behind the lines represent the distributions of the accuracy values in the different runs. On the top plot (2w_1000_04, where SLUG succeeded), the lines reveal a sharp increase of accuracy once the key features are found. The distributions reveal a wide dispersion of results, particularly after the initial generations and until more than half of the evolution. This happens mostly because not all runs find the key features around the same time in the evolution, which means that, in any given generation, some runs have already performed the accuracy "jump" while others have not, resulting in wildly different accuracy values. This also explains why, toward the end of the evolution, when most runs have already found the key features, the dispersion decreases.

Another factor that causes dispersion is that, even on a single run, there may occur several jumps in accuracy because the key features may be found and then lost again. And the fact that this may happen (and indeed happens) is a symptom of insufficient feature selection pressure, which

Table 3 Median test overall accuracy of each method on the GAMETES two-way tasks. Best results for each problem are identified in bold. Results with no statistically significant difference from the best are also highlighted in bold

2w_	10_005	10_01	10_02	10_04	100_005	100_01	100_02	100_04	1000_02	1000_04
GP	**0.628**	**0.682**	**0.710**	0.663	0.521	0.535	0.509	0.510	0.502	0.495
M3GP	**0.622**	**0.677**	0.692	**0.796**	0.513	0.637	0.680	0.537	0.490	0.507
M4GP	0.617	**0.675**	0.692	**0.792**	0.561	0.661	0.681	0.759	0.500	0.511
M4GP-E	0.613	0.665	0.699	**0.784**	**0.607**	**0.672**	0.709	**0.781**	0.692	**0.775**
SLUG	**0.629**	**0.682**	**0.710**	**0.797**	**0.617**	**0.681**	**0.722**	**0.777**	**0.720**	0.753
SLUG-L	-	-	-	-	-	-	-	-	**0.720**	**0.757**

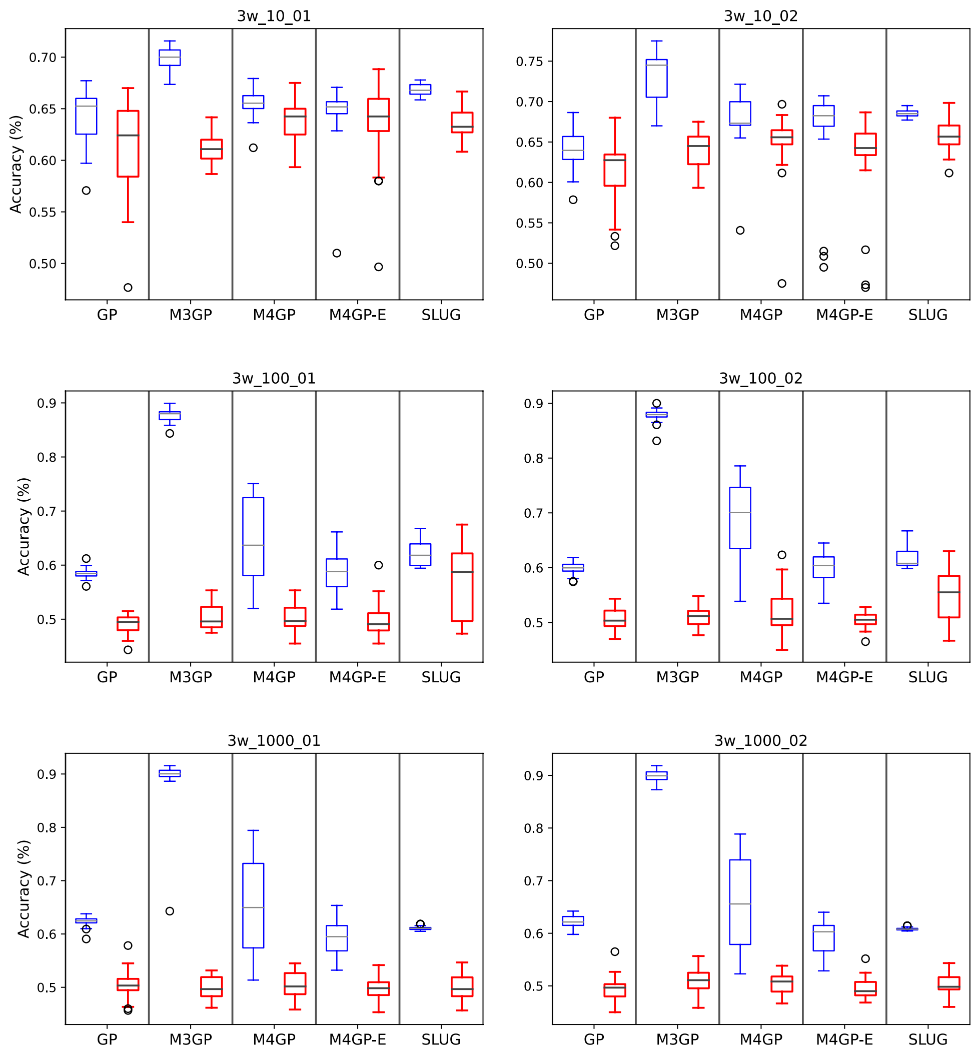


Fig. 6 Performance on the GAMETES three-way datasets. Each plot contains, for each method, the results on the training (left) and test (right) sets

Table 4 Median test overall accuracy of each method on the GAMETES three-way tasks

3w_	10_01	10_02	100_01	100_02	1000_01	1000_02
GP	**0.6241**	0.6275	0.4950	0.5033	**0.5033**	**0.4966**
M3GP	0.6108	0.6450	0.4958	0.5116	**0.4966**	**0.5108**
M4GP	**0.6425**	**0.6558**	0.4966	0.5066	**0.5016**	**0.5083**
M4GP-E	**0.6425**	**0.6425**	0.4908	0.5050	**0.4983**	**0.4900**
SLUG	**0.6375**	**0.6575**	**0.5792**	**0.5550**	**0.4967**	**0.4983**

Best results for each problem are identified in bold. Results with no statistically significant difference from the best are also highlighted in bold

may be the cause for the failure of SLUG that is visible on the bottom plot (3w_1000_02) of Fig. 10.

Feature Selection Pressure

It may seem surprising that, once the key features are found, they may be lost again. It may also be tempting to think that, to avoid this, the GA elitism should guarantee that the best individual is never lost from the population. However, the problem is not lack of elitism, but rather lack of communication between the GA and the GP. Let us recall that the GA informs GP of what features GP can use, and GP informs the GA of what fitness (accuracy) was obtained by the best model. However, GP never reports to the GA which of the allowed features were actually included in the model. Therefore, even if elitism is used to always keep the best GA individual in the population, this individual does not know which of its features are important or not. Each time a new

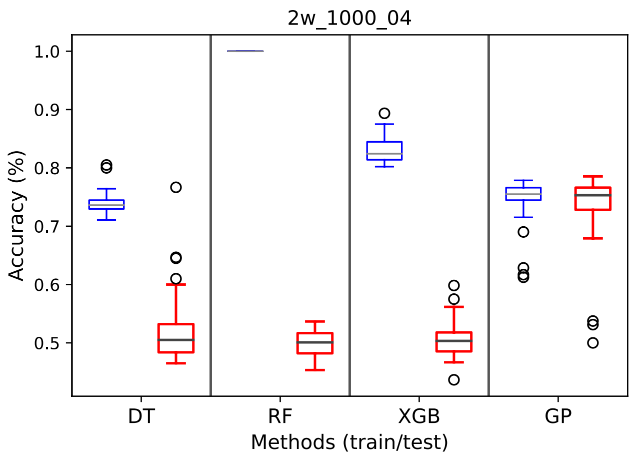


Fig. 7 Performance of different SLUG variants using DT, RF, XGB and GP (the original SLUG) on the GAMETES 2w_1000_04 problem. Each plot contains, for each variant, the results on the training (left) and test (right) sets

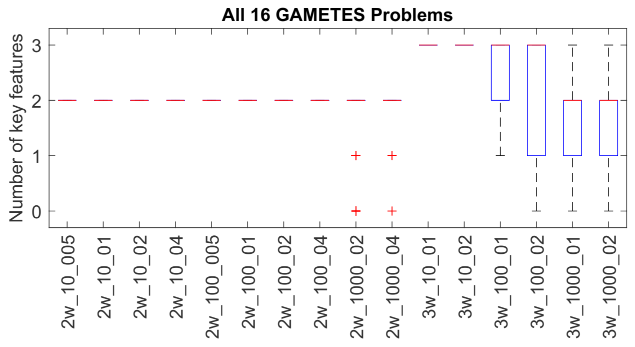


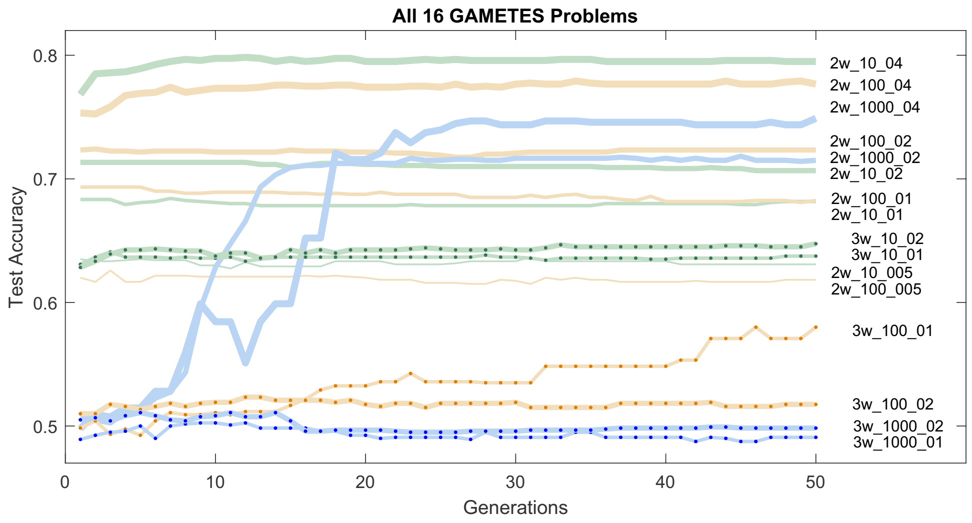
Fig. 9 Boxplot of the number of key features found by the GA at the end of the run, for the 30 runs. Two-way problems have two key features, while three-way problems have three key features

GA individual is evaluated, a new GP run is performed, with new random initial models and a final best model that may or may not be using the right features. The larger the set of features allowed by GA, the higher the probability of GP not finding the important ones.

An enriched communication between GP and the GA is one of the improvements proposed as future work, in the next section. If GP reported back to the GA which subset of allowed features it actually used, the GA could use this information to further reduce the number of allowed features. This would reduce the chances of losing the key features, therefore speeding the convergence toward good models. Ideally, this increased pressure to reduce the number of features would be useful not only to focus on the key features but also to obtain the smallest possible models. When analyzing the high-quality models evolved by the current SLUG system, we realize that many of them contain not only the key features that grant them success, but also extra features that do not seem to bring them any advantage. For example, on the 2w_1000_04 problem (where SLUG succeeded), 26 of the 30 final models contain the two key features, but 23 of them also contain some extra features. Although the (negative) correlation between the number of extra features and the accuracy of the model is not high (−0.38 on training and −0.43 on test), maybe it would still be enough to cause a further reduction of the number of features at the GA level.

What about the cases in which GP is not able to find the key features, like 3w_1000_02? In this case, the problem begins with the GA, that seldom selects the three key features (see Fig. 9), and continues with GP that, in these few cases, is not able to find them among the others. What can GP report back to GA to inform that it should further reduce the number of features? How can GP even know that the reason for not achieving good models is too many allowed features, and not too few? The correlation between the number of features used by the final GP model and its accuracy is practically nonexistent (0.1

Fig. 8 Evolution of test accuracy on all the GAMETES problems. Lines are green/yellow/blue for 10/100/1000-feature problems; lines are thicker for higher signal-to-noise ratios; lines are dotted for the three-way problems. The names of the problems on the right appear by the same order as the lines



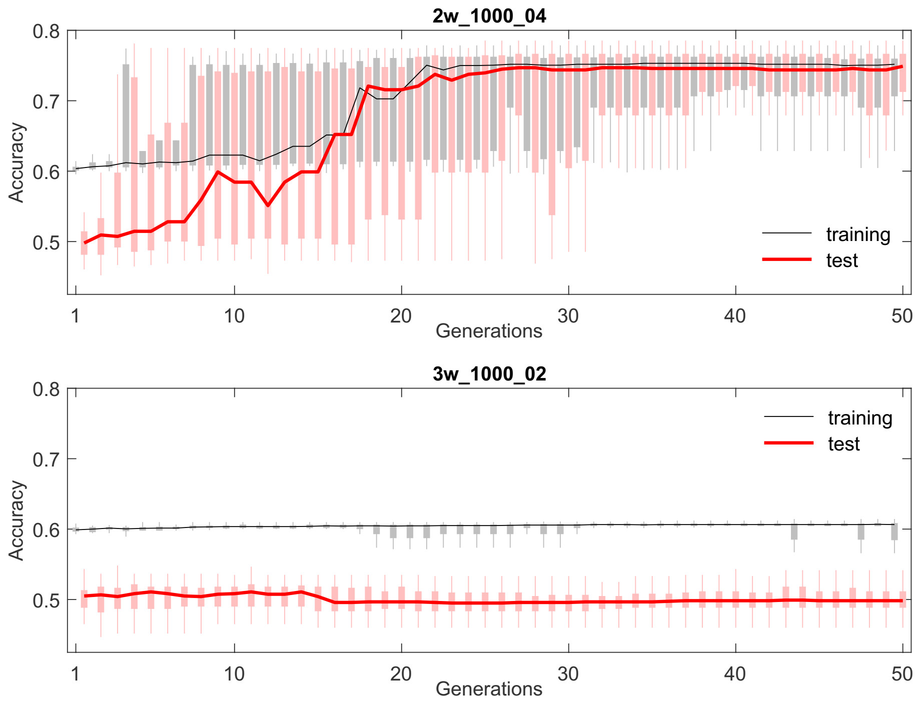


Fig. 10 Evolution of accuracy on the 2w_1000_04 and 3w_1000_02 problems. The lines represent medians of the 30 runs, training (black) and test (red); the lighter boxplots behind them represent the respective distribution of accuracy values

on training and 0.25 on test), so the enriched communication between GP and the GA would not be helpful. Forcing a blind reduction of the number of features at the GA level would be a highly biased decision, since we cannot safely assume that a reduction is always needed, particularly on non-epistatic problems. However, when dealing with epistatic problems like the ones addressed here, it would certainly help. Other possible improvements are mentioned in the next section.

Conclusion and Future Work

We have presented SLUG, a method for feature selection using genetic algorithms (GA) and genetic programming (GP). SLUG implements a cooperative approach that joins these two evolutionary algorithms, where the quality of each GA individual is assessed by performing a GP run with the features selected by the GA. The GA acts like a wrapper, selecting features for GP, the learner. At the end of the process, both the set of GA-selected features and the best GP-induced model are returned, and therefore SLUG comprises the entire pipeline from data preprocessing to predictive modeling. No efforts are put into the optimization of the model, as this is not the main purpose of the work.

We tested SLUG on four regular binary classification datasets and on 16 synthetic datasets produced by GAMETES, a tool for embedding epistatic gene-gene interactions into noisy datasets. We compared the results of SLUG with the ones obtained by standard GP and other GP-based methods like M3GP and two different M4GP variants, one of them also wrapped by the EKF algorithm for feature selection. SLUG obtained the best results in practically all the problems, with special relevance for the good results obtained on the epistatic datasets, whose difficulty was the driver for this research in the first place. Although we maintained the focus on the GAMETES problems, the six new datasets represent a new level of difficulty (finding three key features, instead of two) that, to the best of our knowledge, had not yet been subject to comparative studies. Although most of the new results did not show statistically significant differences between the methods used, the few significant differences revealed an interesting finding: all the methods solve the 10-feature problems; none of the methods solves the 1000-feature problems; only SLUG was able to solve the 100-feature problems.

We discussed the merits and weaknesses of SLUG and the parts that compose it. Its slowness is its obvious limitation, as it requires considerable computational effort to run two nested evolutionary algorithms. We experimented with alternative implementations, replacing the GP backbone of SLUG with faster methods like decision trees, random forests and XGBoost, all wrapped with GA for feature selection. However, even with tuned parameters, none of them was able to catch up with SLUG.

From the above, we conclude that SLUG is a powerful method that performs feature selection while inducing high-quality models, even without putting any effort into model optimization. In the future, we intend to address the main limitation of SLUG, by reducing its computational demands and therefore making it less sluggish. Many other improvements and extensions are possible, like the ones described below.

One possible approach to improve SLUG is by following the approach described in [39] regarding the optimal size of a population. The size of the SLUG populations could be adapted to a calculated number of individuals that can cover all the available features. This would avoid unnecessary computational expense by having smaller populations for low dimensional datasets, and would increase the success rate in high dimensional datasets by improving the coverage rate on the first generation. This is a promising approach, particularly because we know that increasing the GA population size has previously improved the performance of SLUG on the two-way 1000-feature problems [48]. We can hypothesize that increasing the population size may be all that is needed for SLUG to succeed also on the three-way 1000-feature problems.

Regarding other possible improvements, the backbone of SLUG is currently standard GP, which is not appropriate for multiclass classification. However, it can be replaced by other methods. The ones we tried did not produce good models, however other options exist, including M3GP and M4GP themselves, which are some of the best GP-based multiclass classification methods available today. Replacing GP with M3GP would give us the added flexibility of being allowed to plug any learning algorithm to the pipeline to work with the hyper-features evolved by M3GP. Instead of GA+GP, we would have a GA+M3GP+*classifier* pipeline, where GA preselects the features, M3GP uses them to build hyper-features tailored to *classifier*, and *classifier* finally induces an optimized predictive model, with the added advantage that the classifier can be whatever method best suits the needs of the domain application. Naturally, the same rationale can be used for regression instead of classification.

Regarding the improvement of the wrapper, the main issue with GA is, and has always been, the delicate balance between exploration and exploitation, here with an intense concern regarding computational demands. On the one hand, we want to make GA converge faster to a good subset of selected features, also to save computational effort; on the other hand, it must be able to properly explore the search space, particularly on the most difficult higher dimensional problems, but without requiring large populations that would increase the computational time. Alternatively, rather than optimizing the GA as a wrapper, other feature selection methods that show good results on high-dimensional datasets can be explored, e.g., particle swarm optimization [54] or other evolutionary algorithms [55].

To accelerate convergence, GP (or any other backbone SLUG is using) could inform GA of what features are actually being used, from the ones preselected. In case the backbone does not perform feature selection itself, it can probably still inform what features are more important. This way, the GA could use more information from the learner than just the fitness achieved with each subset of features, increasing the cooperation between the two methods. It is reasonable to think that, in this case, the GA binary chromosomes would become real-valued ones, where each bit would now contain a sort of probability of selecting each feature, that the learner could use to build its own models. To promote the exploration of the search space without having to increase the population size, and particularly when adding measures for faster convergence, our idea is to use novelty search [56] on the GA to increase the bias toward yet unexplored subsets of features.

Appendix

See Tables 5, 6 and 7.

Table 5 Holm-corrected p values using Kruskal–Wallis for the regular classification problems. The bold/italic colors indicate that the method on the left is significantly better/worse than the method on the top using $p < 0.05$

HRT	GP	M3GP	M4GP	M4GP-E	SLUG	PRK	GP	M3GP	M4GP	M4GP-E	SLUG
GP	—	0.9922	1.1556	0.2843	*0.0027*	GP	—	*0.0000*	0.7658	0.0627	*0.0441*
M3GP	0.9922	—	0.6884	0.8486	*0.0490*	M3GP	**0.0000**	—	0.0597	0.4882	0.1015
M4GP	1.1556	0.6884	—	0.6599	*0.0124*	M4GP	0.7658	0.0597	—	0.8317	0.7966
M4GP-E	0.2843	0.8486	0.6599	—	0.7852	M4GP-E	0.0627	0.4882	0.8317	—	0.7220
SLUG	**0.0027**	**0.0490**	**0.0124**	0.7852	—	SLUG	**0.0441**	0.1015	0.7966	0.7220	—
ION	GP	M3GP	M4GP	M4GP-E	SLUG	SON	GP	M3GP	M4GP	M4GP-E	SLUG
GP	—	0.3297	2.2914	1.6595	0.3457	GP	—	*0.0000*	*0.0010*	*0.0154*	0.3311
M3GP	0.3297	—	0.9674	0.3734	0.9704	M3GP	**0.0000**	—	0.2653	0.3660	**0.0232**
M4GP	2.2914	0.9674	—	1.7345	1.1531	M4GP	**0.0010**	0.2653	—	0.6295	0.2414
M4GP-E	1.6595	0.3734	1.7345	—	0.3530	M4GP-E	**0.0154**	0.3660	0.6295	—	0.4270
SLUG	0.3457	0.9704	1.1531	0.3530	—	SLUG	0.3311	*0.0232*	0.2414	0.4270	—

Table 6 Holm-corrected p values using Kruskal–Wallis for the two-way GAMETES problems. The bold/italic colors indicate that the method on the left is significantly better/worse than the method on the top using $p < 0.05$

10_005	GP	M3GP	M4GP	M4GP-E	SLUG
GP	—	1.1109	0.0673	**0.0107**	0.9234
M3GP	1.1109	—	0.8996	0.2383	0.5035
M4GP	0.0673	0.8996	—	0.7535	*0.0433*
M4GP-E	*0.0107*	0.2383	0.7535	—	*0.0117*
SLUG	0.9234	0.5035	**0.0433**	**0.0117**	—

10_01	GP	M3GP	M4GP	M4GP-E	SLUG
GP	—	0.2129	0.0701	**0.0194**	0.5589
M3GP	0.2129	—	1.3918	0.5030	0.4243
M4GP	0.0701	1.3918	—	0.9644	0.1472
M4GP-E	*0.0194*	0.5030	0.9644	—	*0.0241*
SLUG	0.5589	0.4243	0.1472	**0.0241**	—

10_02	GP	M3GP	M4GP	M4GP-E	SLUG
GP	—	**0.0000**	**0.0017**	**0.0021**	1.4559
M3GP	*0.0000*	—	0.7673	0.4077	*0.0001*
M4GP	*0.0017*	0.7673	—	0.8110	*0.0020*
M4GP-E	*0.0021*	0.4077	0.8110	—	*0.0109*
SLUG	1.4559	**0.0001**	**0.0020**	**0.0109**	—

10_04	GP	M3GP	M4GP	M4GP-E	SLUG
GP	—	*0.0000*	*0.0000*	*0.0000*	*0.0000*
M3GP	**0.0000**	—	0.8301	0.1779	1.3901
M4GP	**0.0000**	0.8301	—	0.3962	1.3785
M4GP-E	**0.0000**	0.1779	0.3962	—	0.0975
SLUG	**0.0000**	1.3901	1.3785	0.0975	—

100_005	GP	M3GP	M4GP	M4GP-E	SLUG
GP	—	0.3509	0.3911	*0.0012*	*0.0001*
M3GP	0.3509	—	*0.0213*	*0.0000*	*0.0000*
M4GP	0.3911	**0.0213**	—	*0.0000*	*0.0000*
M4GP-E	**0.0012**	**0.0000**	**0.0000**	—	0.3713
SLUG	**0.0001**	**0.0000**	**0.0000**	0.3713	—

100_01	GP	M3GP	M4GP	M4GP-E	SLUG
GP	—	0.9823	*0.0148*	*0.0060*	*0.0001*
M3GP	0.9823	—	0.0599	*0.0072*	*0.0000*
M4GP	**0.0148**	0.0599	—	0.2304	*0.0006*
M4GP-E	**0.0060**	**0.0072**	0.2304	—	0.0796
SLUG	**0.0001**	**0.0000**	**0.0006**	0.0796	—

100_02	GP	M3GP	M4GP	M4GP-E	SLUG
GP	—	0.3664	*0.0040*	*0.0000*	*0.0000*
M3GP	0.3664	—	0.2970	*0.0001*	*0.0000*
M4GP	**0.0040**	0.2970	—	*0.0005*	*0.0000*
M4GP-E	**0.0000**	**0.0001**	**0.0005**	—	*0.0365*
SLUG	**0.0000**	**0.0000**	**0.0000**	**0.0365**	—

100_04	GP	M3GP	M4GP	M4GP-E	SLUG
GP	—	0.0728	*0.0000*	*0.0000*	*0.0000*
M3GP	0.0728	—	0.0539	*0.0000*	*0.0001*
M4GP	**0.0000**	0.0539	—	*0.0015*	*0.0084*
M4GP-E	**0.0000**	**0.0000**	**0.0015**	—	0.1311
SLUG	**0.0000**	**0.0001**	**0.0084**	0.1311	—

1000_02	GP	M3GP	M4GP	M4GP-E	SLUG	SLUG-L
GP	—	0.4726	0.5740	*0.0000*	*0.0000*	*0.0000*
M3GP	0.4726	—	0.0718	*0.0000*	*0.0000*	*0.0000*
M4GP	0.5740	0.0718	—	*0.0000*	*0.0000*	*0.0000*
M4GP-E	**0.0000**	**0.0000**	**0.0000**	—	*0.0003*	*0.0000*
SLUG	**0.0000**	**0.0000**	**0.0000**	**0.0003**	—	0.9174
SLUG-L	**0.0000**	**0.0000**	**0.0000**	**0.0000**	0.9174	—

1000_04	GP	M3GP	M4GP	M4GP-E	SLUG	SLUG-L
GP	—	0.3560	0.0676	*0.0000*	*0.0000*	*0.0000*
M3GP	0.3560	—	0.2035	*0.0000*	*0.0000*	*0.0000*
M4GP	0.0676	0.2035	—	*0.0000*	*0.0000*	*0.0000*
M4GP-E	**0.0000**	**0.0000**	**0.0000**	—	**0.0114**	0.1650
SLUG	**0.0000**	**0.0000**	**0.0000**	*0.0114*	—	0.3762
SLUG-L	**0.0000**	**0.0000**	**0.0000**	0.1650	0.3762	—

Table 7 Holm-corrected p values using Kruskal–Wallis for the three-way GAMETES problems. The bold/italic colors indicate that the method on the left is significantly better/worse than the method on the top using $p < 0.05$

10_01	GP	M3GP	M4GP	M4GP-E	SLUG	10_02	GP	M3GP	M4GP	M4GP-E	SLUG
GP	—	1.0438	0.1143	0.1010	0.1007	GP	—	0.0777	*0.0001*	*0.0190*	*0.0000*
M3GP	1.0438	—	*0.0000*	*0.0003*	*0.0000*	M3GP	0.0777	—	0.0717	1.1082	*0.0346*
M4GP	0.1143	**0.0000**	—	2.1843	0.9175	M4GP	**0.0001**	0.0717	—	0.1721	0.6680
M4GP-E	0.1010	**0.0003**	2.1843	—	1.6145	M4GP-E	**0.0190**	1.1082	0.1721	—	0.1455
SLUG	0.1007	**0.0000**	0.9175	1.6145	—	SLUG	**0.0000**	**0.0346**	0.6680	0.1455	—
100_01	GP	M3GP	M4GP	M4GP-E	SLUG	100_02	GP	M3GP	M4GP	M4GP-E	SLUG
GP	—	0.6412	1.2430	1.2717	*0.0000*	GP	—	1.6806	1.8232	0.8940	*0.0024*
M3GP	0.6412	—	0.9941	1.0946	*0.0005*	M3GP	1.6806	—	1.7649	1.6084	*0.0118*
M4GP	1.2430	0.9941	—	1.5453	*0.0004*	M4GP	1.8232	1.7649	—	1.4744	*0.0288*
M4GP-E	1.2717	1.0946	1.5453	—	*0.0001*	M4GP-E	0.8940	1.6084	1.4744	—	*0.0020*
SLUG	**0.0000**	**0.0005**	**0.0004**	**0.0001**	—	SLUG	**0.0024**	**0.0118**	**0.0288**	**0.0020**	—
1000_01	GP	M3GP	M4GP	M4GP-E	SLUG	1000_02	GP	M3GP	M4GP	M4GP-E	SLUG
GP	—	2.3026	0.9587	1.8317	1.8926	GP	—	0.0810	0.4141	0.9882	0.6128
M3GP	2.3026	—	2.1307	2.4733	1.8938	M3GP	0.0810	—	1.1972	0.0730	0.6951
M4GP	0.9587	2.1307	—	2.0380	2.0859	M4GP	0.4141	1.1972	—	0.5304	1.1277
M4GP-E	1.8317	2.4733	2.0380	—	3.2518	M4GP-E	0.9882	0.0730	0.5304	—	0.7222
SLUG	1.8926	1.8938	2.0859	3.2518	—	SLUG	0.6128	0.6951	1.1277	0.7222	—

Funding Open access funding provided by FCT|FCCN (b-on). This work was partially supported by the FCT, Portugal, through funding of the LASIGE Research Unit (UIDB/00408/2020 and UIDP/00408/2020); MAR2020 program via project MarCODE (MAR$$-$$01.03.01-FEAMP-0047); project AICE (DSAIPA/DS/0113/2019). Nuno Rodrigues and João Batista were supported by PhD Grants 2021/05322/BD and SFRH/BD/143972/2019, respectively; William La Cava was supported by the National Library Of Medicine of the National Institutes of Health under Award Number R00LM012926.

Data availability All datasets used are publicly available. Each dataset has a reference to the location where it can be obtained.

Declarations

Conflict of interest The authors declare no conflicts of interest.

References

1. Cordell HJ. Epistasis: what it means, what it doesn't mean, and statistical methods to detect it in humans. Human Molecular Genetics. 2002;11(20):2463–8. https://doi.org/10.1093/hmg/11.20.2463. http://academic.oup.com/hmg/article-pdf/11/20/2463/1699507/ddf251.pdf
2. Davidor Y. Epistasis variance: A viewpoint on ga-hardness. Foundations of Genetic Algorithms, vol. 1, pp. 23–35. Elsevier; 1991. https://doi.org/10.1016/B978-0-08-050684-5.50005-7
3. Reeves CR, Wright CC. Epistasis in genetic algorithms: An experimental design perspective. In: Proceedings of the 6th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA; 1995. p. 217–224.
4. Altenberg L. B2.7.2. NK fitness landscapes. In: Handbook of Evolutionary Computation, pp. 5–10. IOP Publishing Ltd and Oxford University Press, GBR; 1997.
5. Merz P, Freisleben B. On the effectiveness of evolutionary search in high-dimensional nk-landscapes. In: 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360), 1998; p. 741–745. https://doi.org/10.1109/ICEC.1998.700144
6. Mathias KE, Eshelman LJ, Schaffer JD. Niches in nk-landscapes. In: Martin, W.N., Spears, W.M. (eds.) Foundations of Genetic Algorithms 6, Morgan Kaufmann, San Francisco; 2001. pp 27–46. https://doi.org/10.1016/B978-155860734-7/50085-8
7. Chan KY, Aydin ME, Fogarty TC. An epistasis measure based on the analysis of variance for the real-coded representation in genetic algorithms. In: The 2003 Congress on Evolutionary Computation, 2003. CEC '03., vol. 1, pp. 297–3041. 2003. https://doi.org/10.1109/CEC.2003.1299588
8. Aguirre HE, Tanaka K. Genetic algorithms on nk-landscapes: Effects of selection, drift, mutation, and recombination. In: Cagnoni S, Johnson CG, Cardalda JJR, Marchiori E, Corne DW,

Meyer J-A, Gottlieb J, Middendorf M, Guillot A, Raidl GR, Hart E, editors. Applications of Evolutionary Computing. Berlin, Heidelberg: Springer; 2003. p. 131–42.

9. Pelikan M, Sastry K, Goldberg DE, Butz MV, Hauschild M. Performance of evolutionary algorithms on nk landscapes with nearest neighbor interactions and tunable overlap. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation. GECCO '09, Association for Computing Machinery, New York, NY, USA; 2009. p. 851–858. https://doi.org/10.1145/1569901.1570018
10. Tinós R, Whitley D, Chicano F. Partition crossover for pseudo-boolean optimization. In: Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII. FOGA '15, Association for Computing Machinery, New York, NY, USA; 2015. p. 137–149. https://doi.org/10.1145/2725494.2725497
11. Vanneschi L, Castelli M, Manzoni L. The k landscapes: A tunably difficult benchmark for genetic programming. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation. GECCO '11, Association for Computing Machinery, New York, NY, USA; 2011. p. 1467–1474. https://doi.org/10.1145/2001576.2001773
12. Rochet S. Epistasis in genetic algorithms revisited. Inf Sci. 1997;102(1):133–55. https://doi.org/10.1016/S0020-0255(97)00017-0.
13. Jafari S, Kapitaniak T, Rajagopal K, Pham V-T, Alsaadi F. Effect of epistasis on the performance of genetic algorithms. Journal of Zhejiang University-SCIENCE A. 2018;20. https://doi.org/10.1631/jzus.A1800399
14. Lee J, Kim Y-H. Epistasis-based basis estimation method for simplifying the problem space of an evolutionary search in binary representation. Complexity. 2019;13:2095167.
15. Ansarifar J, Wang L. New algorithms for detecting multi-effect and multi-way epistatic interactions. Bioinformatics. 2019;35(24):5078–85. https://doi.org/10.1093/bioinformatics/btz463. http://academic.oup.com/bioinformatics/article-pdf/35/24/5078/31797822/btz463.pdf
16. Mo H, Li Z, Zhu C. A kind of epistasis-tunable test functions for genetic algorithms. Concurrency and Computation: Practice and Experience. 2021;33(8):5030. https://doi.org/10.1002/cpe.5030. e5030 cpe.5030.
17. Hall MA. Correlation-based feature selection for machine learning. PhD thesis. 1999.
18. Nazareth DL, Soofi ES, Zhao H. Visualizing attribute interdependencies using mutual information, hierarchical clustering, multidimensional scaling, and self-organizing maps. In: 2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07), 2007. p. 53–53. https://doi.org/10.1109/HICSS.2007.608
19. Korns MF. Genetic programming symbolic classification: A study. In: Banzhaf W, Olson RS, Tozier W, Riolo R, editors. Genetic Programming Theory and Practice XV. Cham: Springer; 2018. p. 39–54.
20. Petinrin OO, Wong K-C. Protocol for epistasis detection with machine learning using genepi package. Methods Mol Biol. 2021;2212:291–305.
21. Urbanowicz RJ, Kiralis J, Sinnott-Armstrong NA, *et al.* GAMETES: a fast, direct algorithm for generating pure, strict, epistatic models with random architectures. BioData Mining, 2012;5(16). https://doi.org/10.1186/1756-0381-5-16
22. La Cava W, Silva S, Danai K, Spector L, Vanneschi L, Moore JH. Multidimensional genetic programming for multiclass classification. Swarm Evol Comput. 2019;44:260–72. https://doi.org/10.1016/j.swevo.2018.03.015.
23. Sohn A, Olson RS, Moore JH. Toward the automated analysis of complex diseases in genome-wide association studies using genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference. GECCO '17, Association for Computing Machinery, New York, NY, USA; 2017. p. 489–496. https://doi.org/10.1145/3071178.3071212
24. Urbanowicz RJ, Meeker M, La Cava W, Olson RS, Moore JH. Relief-based feature selection: Introduction and review. J Biomed Inform. 2018;85:189–203. https://doi.org/10.1016/j.jbi.2018.07.014.
25. Lanzi PL. Fast feature selection with genetic algorithms: a filter approach. In: Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97), 1997. p. 537–540. https://doi.org/10.1109/ICEC.1997.592369
26. Chaikla N, Qi Y. Genetic algorithms in feature selection. In: IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028), vol. 5, pp. 538–5405. 1999. https://doi.org/10.1109/ICSMC.1999.815609
27. Li A-D, Xue B, Zhang M. Multi-objective feature selection using hybridization of a genetic algorithm and direct multisearch for key quality characteristic selection. Inf Sci. 2020;523:245–65. https://doi.org/10.1016/j.ins.2020.03.032.
28. Hussein F, Kharma N, Ward R. Genetic algorithms for feature selection and weighting, a review and study. In: Proceedings of Sixth International Conference on Document Analysis and Recognition, 2001. p. 1240–1244. https://doi.org/10.1109/ICDAR.2001.953980
29. Xue B, Zhang M, Browne WN, Yao X. A survey on evolutionary computation approaches to feature selection. IEEE Trans Evol Comput. 2016;20(4):606–26. https://doi.org/10.1109/TEVC.2015.2504420.
30. Lavine BK, White CG. Boosting the performance of genetic algorithms for variable selection in partial least squares spectral calibrations. Appl Spectrosc. 2017;71(9):2092–101.
31. Chiesa M, Maioli G, Colombo GI, *et al.* Gars: Genetic algorithm for the identification of a robust subset of features in high-dimensional datasets. BMC Bioinformatics, 2020;21(54). https://doi.org/10.1186/s12859-020-3400-6
32. Wutzl B, Leibnitz K, Rattay F, Kronbichler M, Murata M, Golaszewski SM. Genetic algorithms for feature selection when classifying severe chronic disorders of consciousness. PLoS ONE. 2019;14(7):1–16. https://doi.org/10.1371/journal.pone.0219683.
33. Shin K-S, Lee Y-J. A genetic algorithm application in bankruptcy prediction modeling. Expert Syst Appl. 2002;23:321–8.
34. Seo K-K. Content-based image retrieval by combining genetic algorithm and support vector machine. In: de Sá JM, Alexandre LA, Duch W, Mandic D, editors. Artificial Neural Networks - ICANN 2007. Berlin, Heidelberg: Springer; 2007. p. 537–45.
35. Chion C, Landry J-A, Costa LD. A genetic-programming-based method for hyperspectral data information extraction: Agricultural applications. IEEE Trans Geosci Remote Sens. 2008;46(8):2446–57. https://doi.org/10.1109/tgrs.2008.922061.
36. García-Dominguez A, Galván-Tejada CE, Zanella-Calzada LA, Gamboa-Rosales H, Galván-Tejada JI, Celaya-Padilla JM, Luna-García H, Magallanes-Quintanar R. Feature selection using genetic algorithms for the generation of a recognition and classification of children activities model using environmental sound. Mobile Information Systems Article ID. 2020;8617430:12. https://doi.org/10.1155/2020/8617430.
37. Muñoz L, Silva S, Trujillo L. M3gp - multiclass classification with gp. In: EuroGP. 2015.
38. Smith MG, Bull L. Feature construction and selection using genetic programming and a genetic algorithm. In: Ryan C, Soule T, Keijzer M, Tsang E, Poli R, Costa E, editors. Genetic Programming. Berlin, Heidelberg: Springer; 2003. p. 229–37.
39. Langdon WB, Harrison AP. GP on SPMD parallel graphics hardware for mega bioinformatics data mining. Soft

Comput. 2008;12(12):1169–83. https://doi.org/10.1007/s00500-008-0296-x.

40. Doucette JA, McIntyre AR, Lichodzijewski P, Heywood MI. Symbiotic coevolutionary genetic programming: a benchmarking study under large attribute spaces. Genet Program Evolvable Mach. 2012;13:71–101.
41. Rodriguez-Coayahuitl L, Morales-Reyes A, Escalante HJ, Coello Coello CA. Cooperative co-evolutionary GP for high dimensional problems. In: Baeck, T., Preuss, M., Deutz, A., Wang2, H., Doerr, C., Emmerich, M., Trautmann, H. (eds.) 16th International Conference on Parallel Problem Solving from Nature, Part II. LNCS, vol. 12270, Springer, Leiden, Holland; 2020. p. 48–62. https://doi.org/10.1007/978-3-030-58115-2_4
42. Guyon I, Gunn S, Hur AB, Dror G. In: Guyon, I., Nikravesh, M., Gunn, S., Zadeh, L.A. (eds.) Design and Analysis of the NIPS2003 Challenge, Springer, Berlin, Heidelberg; 2006. p. 237–263. https://doi.org/10.1007/978-3-540-35488-8_10
43. Rodrigues NM, Batista JE, Silva S. Ensemble genetic programming. In: Hu T, Lourenço N, Medvet E, Divina F, editors. Genetic Programming. Cham: Springer; 2020. p. 151–66.
44. Batista JE, Rodrigues NM, Cabral AIR, Vasconcelos MJP, Venturieri A, Silva LGT, Silva S. Optical time series for the separation of land cover types with similar spectral signatures: cocoa agroforest and forest. Int J Remote Sens. 2022;43(9):3298–319. https://doi.org/10.1080/01431161.2022.2089540.
45. Pei W, Xue B, Shang L, Zhang M. New fitness functions in genetic programming for classification with high-dimensional unbalanced data. In: 2019 IEEE Congress on Evolutionary Computation (CEC), IEEE; 2019. p. 2779–2786.
46. Tran B, Xue B, Zhang M. Genetic programming for feature construction and selection in classification on high-dimensional data. Memetic Computing. 2015;8(1):3–15. https://doi.org/10.1007/s12293-015-0173-y.
47. Silva S, Tseng Y-T. Classification of seafloor habitats using genetic programming. In: Lecture Notes in Computer Science, Springer; 2008. p. 315–324. https://doi.org/10.1007/978-3-540-78761-7_32
48. Rodrigues NM, Batista JaE, La Cava W, Vanneschi L, Silva S. Slug: Feature selection using genetic algorithms and genetic programming. In: Genetic Programming: 25th European Conference, EuroGP 2022, Held as Part of EvoStar 2022, Madrid, Spain, April 20–22, 2022, Proceedings, Springer, Berlin, Heidelberg; 2022. p. 68–84. https://doi.org/10.1007/978-3-031-02056-8_5
49. Dua D, Graff C. UCI Machine Learning Repository. 2017. http://archive.ics.uci.edu/ml
50. Zhang S. sonar.all-data. Kaggle. 2018. https://www.kaggle.com/ypzhangsam/sonaralldata
51. Urbanowicz RJ, Kiralis J, Sinnott-Armstrong NA, Heberling T, Fisher JM, Moore JH. Gametes: a fast, direct algorithm for generating pure, strict, epistatic models with random architectures. BioData Mining. 2012;5:16–16.
52. Kononenko I. Estimating attributes: Analysis and extensions of relief. In: ECML. 1994.
53. 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, Duchesnay E. Scikit-learn: Machine learning in Python. J Mach Learn Res. 2011;12:2825–30.
54. Tran B, Xue B, Zhang M. Variable-length particle swarm optimization for feature selection on high-dimensional classification. IEEE Trans Evol Comput. 2019;23(3):473–87. https://doi.org/10.1109/tevc.2018.2869405.
55. Shang Y, Zheng X, Li J, Liu D, Wang P. A comparative analysis of swarm intelligence and evolutionary algorithms for feature selection in SVM-based hyperspectral image classification. Remote Sensing. 2022;14(13):3019. https://doi.org/10.3390/rs14133019.
56. Lehman J, Stanley KO. Exploiting open-endedness to solve problems through the search for novelty. In: Bullock, S., Noble, J., Watson, R.A., Bedau, M.A. (eds.) Proceedings of the Eleventh International Conference on Artificial Life (Alife XI), MIT Press, Cambridge, MA; 2008. p. 329–336.