Enhancing Discrimination Power with Genetic Feature Construction: A Grammatical Evolution Approach

Patrícia Miquilini*, Rodrigo C. Barros† Vinícius V. de Melo* and Márcio P. Basgalupp*
*Universidade Federal de São Paulo
Instituto de Ciência e Tecnologia, São José dos Campos, SP, Brazil
Emails: {patricia.miquilini,vinicius.melo,basgalupp}@unifesp.br
†Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática, Porto Alegre, RS, Brazil
Email: rodrigo.barros@pucrs.br

***Abstract*—Data set preprocessing is a critical step for the successful application of machine learning algorithms in classification tasks. Even though we rely on learning algorithms to pinpoint the optimal decision boundaries in the feature space by properly detecting latent relationships among the input features, their performance is often bounded by the discriminative power of the available features. Therefore, much effort has been devoted to developing preprocessing methods that are capable of transforming the input data with the final goal of aiding the machine learning algorithm in building high-quality classification models. One such a method is feature construction, which is a flexible preprocessing procedure that exploits linear and nonlinear transformations of the original feature space in an attempt to capture useful information that is not explicit in the original data. Since the task of feature construction can be modelled as a heuristic search in the space of novel latent features, this paper investigates an evolutionary approach for performing such a task, namely grammatical evolution (GE). In our proposed approach, GE is employed for building an extra novel feature from the available input data in order to maximize the predictive performance of the learning algorithm in training data. Results show that many interesting implicit relationships are indeed found by the evolutionary approach, improving the performance of two well-known decision-tree induction algorithms.**

I. Introduction

Machine Learning algorithms are said to learn from experience by building either predictive or descriptive models based on the discovery of hidden relationships in the available data. Most algorithms require the data to be structured in a feature-value fashion, the so-called data sets. A data set consists of a collection of objects, which are real-world entities described by a set of features (characteristics, attributes) often accompanied by a given label (class). Such a label is useful for organizing the objects into meaningful categories, and defining which object should be assigned to which label is a well-known learning problem called classification.

Arguably the most studied task in machine learning, classification can be seen as the task of formally defining an objective function capable of mapping the objects from the application domain into their corresponding labels based on previously-labeled data, commonly regarded as the training set. The performance of a learning algorithm is heavily depended on the training set that is used to build the predictive model.

For the cases in which the set of available features is not enough for the learning algorithm to generate a model with satisfactory accuracy, the specialized literature suggests the class of constructive induction algorithms [1], whose goal is to transform the original space into a novel space of features with increased discriminatory power. Several systems have been developed over the years to perform constructive induction [1]–[9].

The method of constructing a feature is complex, often involving the selection of existing features followed by multiple transformations [5]. It is a costly, time-consuming, hand-crafted work frequently unfeasible due to resource constraints. With that being said, the paradigm of evolutionary algorithms has been successfully employed for the automatic generation of programs and functions [5], [10]–[16], mapping a given problem into an automatic heuristic search that is guided by the principles of evolution. Grammatical Evolution (GE), in particular, arises as a well-suited approach for such tasks. It is an evolutionary algorithm in which computer programs can be generated in any programming language through genetic evolution aided by a context-free grammar [17], [18]. GE combines the simplicity of representation from genetic algorithms and the flexibility of grammar-based genetic programming in order to perform a robust search in the space of candidate solutions.

In this paper, we propose handling the feature construction task with GE. Our goal is to build a *single* novel feature to a given data set using GE, based on a context-free grammar that allows for the combination of previously-existent features. Multiple distinct features could be easily created by executing multiple runs of our proposed approach, but here we investigate whether any substantial improvement can be obtained with a single automatically-constructed feature. Our main motivation is based on the success achieved by previous work on the area [5], [6].

Our approach employs the *wrapper* strategy within its evolutionary cycle, i.e., we evaluate the quality of each candidate solution with respect to the execution of two well-known decision-tree induction algorithms: C4.5 [19] (its Java-based version, J48 [20]) and REPTree [20] (similar to C4.5 but with reduced-error pruning). After the evolutionary process carried out by GE, the classification accuracy of the data set with the new feature is evaluated and compared with the results provided by the classification model built from the original data set.

In order to assess the predictive performance of the proposed approach, we test it on a variety of public data sets from the UCI machine learning repository [21]. We perform an extensive experimental analysis that clearly indicates the benefits of including a single novel feature in the data set, paving the way for the broad application of GE-based feature construction in several application domains.

The remaining of this paper is organized as follows. Section II describes work related to the construction of features for classification tasks. Section III introduces basic concepts of Grammatical Evolution. Section IV presents our novel evolutionary approach to build novel features based on linear and nonlinear combinations of the existent features. Section V reports the experimental analysis that was performed to validate the results along a discussion of our findings. Finally, Section VI ends this paper with our conclusions and future work directions.

II. Related Work

Real-world data sets may have features that do not have sufficient discriminatory power to generate reasonable decision boundaries in classification problems [22]. Thus, the research community has been focused on constructive induction algorithms whose goal is to transform the original feature space into a novel space that allows for more precise classification [1]. Such a task is referred to as *feature construction* or *feature generation*.

To construct features for classification tasks, evolutionary algorithms such as Genetic Programming [23] have been extensively explored [10]–[12]. These algorithms can build programs and expressions dynamically, and are specially useful in problems when the space of features has a high dimension. Guo [10] proposed a genetic program approach to generate features and used those as input to a neural classifier with the goal of identifying six bearing conditions. Krawiec [11] also proposed a genetic programming approach to build features in order to improve machine learning classifiers. Neshatian [12] employed genetic programming to construct novel features from the original features and used decision-tree classifiers to analyze their efficiency. More recently, Ahmed [24] proposed a method to build multiple features based on feasible subtrees in the best individual. The previously mentioned studies make use of genetic programming to build novel feature(s) and their results show that the data sets with the novel feature(s) improves the classification accuracy of the tested classifiers.

Gavrilis [5] proposed a method for selecting and building features using Grammatical Evolution and neural networks. The selection process chooses features from the original set to build novel features, and only those are then used by the classifier. The context-free grammar used to build new features employ algebraic expressions like: multiplication, subtraction, division, sine, cosine, log, and exponential. Experiments were performed with three different classifiers: $RBF\text{-}NN$ and $MLP\text{-}NN$ (for regression problems) and $K\text{-}NN$ (for classification problems). The proposed algorithm indeed improved the performance of classifiers when new features were built.

III. Grammatical Evolution

Grammatical Evolution (GE) algorithms are defined as efficient methods for automatic building programs (and functions) using context-free grammars. GE employs a robust mapping process, where from simple binary sequences it is possible to produce a code in any programming language.

As an Evolutionary Algorithm, GE is inspired by biological phenomena and can be related to the generation process of a protein of the genetic material of an organism [18].

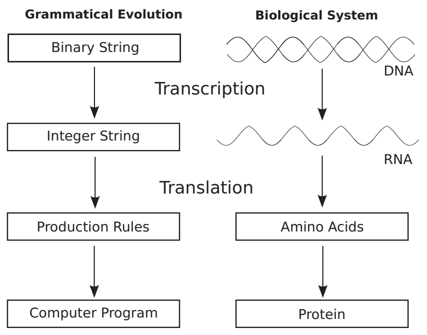

Fig. 1. Comparison between the process of mapping the Grammatical Evolution and biological organisms [18].

Figure 1 shows the comparison between the process of mapping in GE and in biological organisms. The analogy is observed since the beginning of the process in which DNA is equivalent to binary strings in GE. In the generation of protein, we need to slightly change the format: DNA turns into RNA and binary strings turn into strings of integers. From this new sequence, the translation is performed, RNA or string of integers, to amino acid sequence or production rules, respectively. The results from the process are generated and on biological systems we had the phenotype and in GE we had the function.

GE needs a grammar to convert the string individual into a code, evolutionary operators to select and modify the individuals, and a fitness function to evaluate such individuals. The next section introduces these procedures applied for constructing features.

IV. Constructing Features

The construction of the new feature can be summarized in four steps:

1) The input data set is divided into two sets: training and testing;
2) After the division, the evolutionary process starts. Each individual of the population is a solution mapped by the grammar (IV-A), to calculate the new feature. For each individual in the population decoded into a function, one builds a classifier on the training set of examples;
3) The individual's fitness is the mean of the cross-validation (IV-C);
4) The best individual in the population is selected and used for evaluating the test set. The process returns the best function to construct a new feature and the number of correct predictions associated with it.

The following sections will present the main features of Grammatical Evolution as: grammar (IV-A), genetic operators (IV-B) and the fitness function (IV-C).

A. Grammar

A context-free grammar expresses a language made up of production rules. Context-free grammars are composed of terminal symbols: items that can appear in a language, and non-terminal symbols: items that can be expanded into one or more non-terminal or terminal symbols [17]. Grammars are used according to their definition in the GE context; they describe the output language to be produced by the system. Therefore, it is mandatory to have an appropriate grammar capable of generating useful solutions [18].

Figure 2 illustrates the grammar used in this work in order to generate functions, which can be from a single original attribute to many different attributes resulted by a set of very simple operators and real values.

The symbols of this grammar are detailed as follows.

- att: randomly select an attribute from the data set, returning a vector with the values.
- +: the sum of a real number to the elements of the vector.
- ++: the sum of two vectors.
- *: the multiplication a real number to the elements of the vectors.
- **: the multiplication of two vectors.
- sqrt: the square root of the elements of the vector or a real number. When applied over negative numbers, the function returns 0.
- Number: the selection of a random number from 1 to the number of features of the data set, except the last feature, the classe attribute.
- RealNumber: the selection of a random real number.

$\langle newAtt \rangle$ ::= $\langle vector \rangle$

$\langle vector \rangle$ ::= (att $\langle numAtt \rangle$) | (+ $\langle number \rangle$ $\langle vector \rangle$) | (++ $\langle vector \rangle$ $\langle vector \rangle$) | (* $\langle number \rangle$ $\langle vector \rangle$) | (** $\langle vector \rangle$ $\langle vector \rangle$) | (sqrt $\langle vector \rangle$)

$\langle numAtt \rangle$ ::= (Number)

$\langle number \rangle$ ::= (RealNumber) | (sqrt $\langle number \rangle$)

Fig. 2. Grammar for constructing a new feature.

It is important to mention that we have used the name $< vector >$ to represent an attribute or a function resulting an attribute. For example, the operation $**$ is not a real vector multiplication (which would result in a number), but a multiplication of each element of two vectors. I.e., $< v_1 ** v_1 >$ results a new vector v_{new} where $v_{new}[i] = v_1[i] * v_1[i]$, for all instances i. In practice, it is an easier way to represent what a new attribute really means, since we have to apply it over all instances from the data set.

The grammar presented above generates expressions such as (i) $++(att(2))(att(3))$ and (ii) $**(att(0))(sqrt(att(3)))$. In practice, these two examples of individuals would represent, respectively, the construction of the following two new attributes: (i) $att_{new}[i] = att_2[i] * att_3[i]$ for all instances i; and (ii) $att_{new}[i] = att_0[i] * \sqrt{att_3[i]}$, for all instances i.

B. Genetic Operators

The evolutionary process starts with a random initial population. The individuals can have different lengths, with a minimum size of ten codons. In order to evolve from this initial population, the following independents evolutionary operators can be applied: mutation, crossover, and duplication, each having its probability of application, respectively 10%, 85% and 5%. Operators should be applied until all individuals for the new population are generated.

For crossover process two individuals are selected by tournament; then, they pass through the crossover operation of one point, where a position is selected at the individuals and combining half of the first parent and half of the other, generating two children.

The mutation process involves the selection of an individual with the tournament. This individual will be traversed, gene-by-gene, where each position has a probability of having its value replaced by a random value.

The duplication process selects an individual by tournament. Two positions of its genotype are randomly selected and all the elements between them are copied to the end of the individual.

The new population is generated by selecting individuals from the original population and applying the genetic operators. The new population is formed by children from the crossover process and we maintained two of the best individuals from the generation before (elitism).

C. Fitness Function

To measure the quality of individuals, the classifier built on the training set is evaluated on the validation set. The performance of the classifiers is measured by the amount of correct predictions in the validation set. The method used to evaluate the accuracy was the 5-fold cross-validation.

In Figure 3, we can observe how the fitness evaluation of a new attribute occurs. First, a given individual is mapped into its corresponding attribute, and then it is incorporated into the original training set. Next, the training set is partitioned into a training set and a validation set by using a 5-fold cross-validation technique [25]. The term "validation set" is used in here instead of "test set" to avoid confusion with the test set (which is not used during the grammatical evolution), and also due to the fact that we are using the performance measure of a candidate attribute on those validation sets to guide the evolutionary search for a better function. The same cannot be done with test sets, which are exclusively used for assessing the predictive performance of a decision tree generated from the data set with the new attribute.

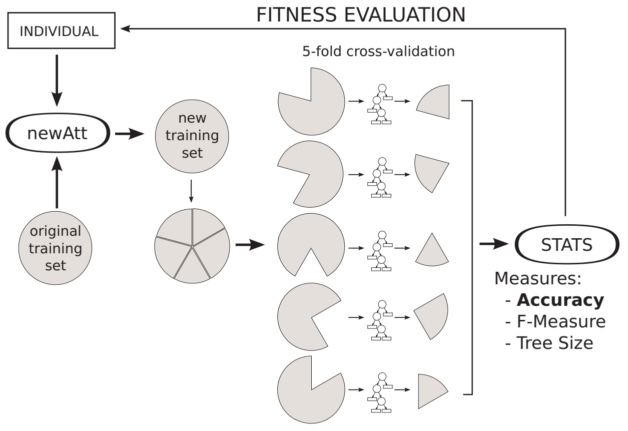

Fig. 3. Fitness function evaluation process.

After dividing each training set into "sub-training" (4/5 pie) and "validation" (1/5 pie), we induce a decision tree for each sub-training set available. For evaluating the predictive performance of these decision trees, we use the corresponding validation sets. Statistics regarding the predictive performance and the size of each decision tree are recorded (*e.g.*, accuracy, F-Measure, precision, recall, total number of nodes/leaves, etc.), and can be used individually or combined as the fitness function. In this work, the fitness is the average Accuracy; the decision tree induction algorithms are the J48 (version of C4.5 [19]) and REPTree, both well-know methods available in the free machine learning tool Weka [20].

D. Addressing special issues

While evolving new attributes by combining other ones, we have to deal with at least two special situations: nominal attributes and missing values. In this work, we address these issues according to the following strategies.

Nominal attributes are converted into numerical attributes by using the index of each nominal value instead of its original value. This is a simple conversion which is useful for our investigation, but not necessarily the best one.

When faced with missing values in a particular instance, our GE-based approach uses a very simple strategy: it replicates the missing value for the new attribute (only for the corresponding instance). Since the focus of our approach is to generate a new feature and not an imputation method, we decided to let the classifier deal with it, i.e., for the evolutionary process decide when it is worth it or not to use features with missing values in the new attribute.

Dealing with such issues was an essential step in the construction of our GE-based approach as is not limited in dealing only with both numeric complete (without missing values) data sets. It is important to mention, however, that despite dealing with both numerical and nominal attributes, the new attribute – evolved by our GE-based approach – is always numerical.

E. Contributions

As we have discussed in the related work section, there are other evolutionary approaches – mostly GP ones – for selecting and generating features in the context of machine learning. However, this work consistently differs from the previous work in several aspects, which are presented as follows.

- In [5] the authors tested a few data sets having two artificial neural networks and k-NN as classifiers. In this work, we employ two popular decision tree induction algorithms, many data sets, a simpler grammar, and a single feature to investigate if an improvement could yet be achieved;

- In this work, each individual represents only one new attribute instead of a set of new attributes. It means that, in practice, our individual can be smaller; consequently, less time-consuming when applying the genetic operators. It is also unnecessary any additional chromosome representation, as in [5];

- Once we intend to create/evolve a single attribute, the idea is to use it in conjunction with the original data set instead of "replacing" it. This choice is based on a very reasonable explanation. Our objective is neither feature selection nor dimensionality reduction, but feature construction. And not any construction, but a new and non-redundant attribute, and, mainly, an attribute capable of adding a discriminatory information that the classifier is unable to detect or treat when using the original data set. A redundant attribute would not add useful information to the data set, it would not change the classifier's performance, and consequently the fitness value. We can use the same idea if we think about a relationship/pattern between attributes that would be easily extracted by a classifier from the original data set (a simple sum of two attributes, for example). Again, this single relationship would not change the classifier's performance either. Then, we focus the search on a unique, non-redundant, and representative attribute for the classification task;

- Among several classification methods available in the literature, in this work we have chosen the well-known and widely-used decision tree induction algorithms, which represent one of the most popular techniques for dealing with classification problems. This choice was based on their main advantages, such as (i) the induction of DTs does not require any domain knowledge; (ii) DT induction algorithms can handle high-dimensional data; (iii) the representation of discovered knowledge in tree form is intuitive and easy to be assimilated by humans; and (iv) the learning and classification steps of DT induction are simple and fast [26]. Given that our approach is an evolutionary algorithm, which we know could be very time-consuming depending on the classifier, we decided to use a powerful one (not as much as a neural network) and fast (not as much as a k-NN). In addition, another advantage of using decision trees is the ability of the induction algorithms - in particular the top-down and recursive partitioning ones - in "automatically" dealing with redundant attributes. Then, this choice strengthens the idea of generating a non-redundant attribute;

- In this work, the GE-based approach is not restricted to data sets entirely composed of numerical attributes, since there is a very simple way to dealing with nominal attributes, as above-mentioned in Section IV-D.

Therefore, to the best of our knowledge, this is the first work to propose a GE-based approach for automatically constructing single and non-redundant attributes to add completely new information to classification data sets, with both numerical and nominal attributes.

V. Experimental Analysis

In this section, we present the methodology employed during the empirical analysis. First, we present in Section V-A the public classification data sets used to evaluate the effectiveness of our approach in automatically generating new and non-redundant attributes. In Section V-B we provide the parameters used in the grammatical evolution algorithm, whereas the statistical analysis process is described in Section V-C. Finally, in Section V we present the results and discussion about the comparative analysis among our approach and the baselines.

A. Data sets

We have applied the proposed GE-based approach for feature construction to many classification data sets collected from the UCI machine learning repository [21]: anneal, arrhythmia, audiology, bridges_version1, car, cylinder_bands, glass, iris, kdd_synthetic_control, segment, semeion, sick, tep.fea, vowel, winequality_red and winequality_white. Table I summarizes these data sets by presenting important information such as number of instances (# instances), number of numeric (# numeric) and nominal (# nominal) attributes, percentage of missing values (% missing) and the number of classes (# classes).

TABLE I. Summary of the 16 data sets used in the experiments.

data set	# instances	# numeric	# nominal	% missing	# classes
anneal	898	6	32	0.00	6
arrhythmia	452	206	73	0.32	16
audiology	226	0	69	2.03	24
bridges_version1	107	3	9	5.53	6
car	1728	0	6	0.00	4
cylinder_bands	540	18	21	4.74	2
glass	214	9	0	0.00	7
iris	150	4	0	0.00	3
kdd_synthetic	600	60	1	0.00	6
segment	2310	19	0	0.00	7
semeion	1593	265	0	0.00	2
sick	3772	6	22	5.54	2
tep.fea	3572	7	0	0.00	3
vowel	990	10	3	0.00	11
winequality_red	1599	11	0	0.00	10
winequality_white	4898	11	0	0.00	10

B. Parameters

Table II shows the user-defined parameter values used in our GE algorithm, which is available in the ECJ framework, a Java-based Evolutionary Computation Research System [27]. The parameter values were based on our previous experience in using evolutionary algorithms, and we have made no attempt to optimize them; this is a topic left for future research. *Max number of generations* is the algorithm's stopping criteria. Due to the fact that GE is a non-deterministic technique, we have run it 5 times for each one of the 10 training/test set folds generated by the 10-fold cross-validation procedure. After running GE over the data sets presented in Table I, we have calculated the average and standard deviation of the 5 executions for each fold and measure, and then the average of the ten folds. This procedure was applied for both versions: GE-J48, which uses J48 as a classifier during the evaluation, and GE-REPTree, which uses REPTree as DT-induction algorithm to compute the fitness function. Considering J48 and REPTree applied to the original data set, we have calculated the averages and standard deviations for a single run with ten folds as these algorithms are deterministic.

TABLE II. Configurable GE parameters.

Parameter	Value
Initialization Probability	85%
Number of Individuals	200
Minimum Individual Size	10
Maximum Individual Size	50
Number of Generations	50
Crossover Probability	90%
Duplication Probability	5%
Mutation Probability	5%
Tournament Size	7
Elite	2

C. Statistical Analysis

In order to evaluate the statistical significance of the experimental results, we present the results of hypothesis tests by following the well-known approach proposed by Demšar [28]. In brief, this approach seeks to compare multiple algorithms on multiple data sets based on the use of the Friedman test followed by a corresponding post-hoc test, Nemenyi. The Friedman test is a non-parametric counterpart of ANOVA, and works as follows. Let R_i^j be the rank of the j^{th} of k algorithms on the i^{th} of N data sets, the Friedman test compares the average ranks of these k algorithms, i.e., $R_j = \frac{1}{N}\sum_i R_i^j$. Then, the Friedman statistic, given by:

$$\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right] \tag{1}$$

is distributed according to χ_F^2 with $k-1$ degrees of freedom, when N and k are large enough.

Since Iman and Davenport [29] showed that Friedman's χ_F^2 is undesirably conservative, they derived an adjusted statistic:

$$F_f = \frac{(N-1)\times\chi_F^2}{N\times(k-1)-\chi_F^2}, \tag{2}$$

which is distributed according to the F-distribution with $k-1$ and $(k-1)(N-1)$ degrees of freedom.

If the null hypothesis of similar performances is rejected, we proceed with the Nemenyi post-hoc test for pairwise comparisons. We can say that the performance of two classifiers is significantly different if their corresponding average ranks differ by at least the critical difference

$$CD = q_\alpha\sqrt{\frac{k(k+1)}{6N}}, \tag{3}$$

where critical values q_α are based on the Studentized range statistic divided by $\sqrt{2}$.

D. Results

Table III shows the predictive accuracy values for GE-J48, J48, GE-REPTree and REPTree. It presents the average accuracy over a 10-fold cross-validation procedure. Observe that the decision-trees induced over the new data sets – with the additional attributes added by our GE-based approach – generates trees with the best accuracy values in 12 out of 16 data sets. In three cases – anneal, semeion, and tep.fea – more than one method presented the best accuracy, then we highlighted in bold the one with the smallest standard deviation.

TABLE III. ACCURACY OF THE DECISION TREES GENERATED BY THE DT-INDUCTION ALGORITHMS FROM DATA SETS WITH (GE-J48 AND GE-REPTREE) OR WITHOUT (J48 AND REPTREE) THE NEW ATTRIBUTES. STANDARD DEVIATIONS ARE SHOWN BELOW THE ACCURACY VALUES, BETWEEN BRACKETS.

data set	GE-J48	J48	GE-REPTree	REPTree
anneal	**0.986**	0.986	0.980	0.979
	(0.010)	(0.012)	(0.016)	(0.017)
arrhythmia	0.655	0.651	**0.670**	0.670
	(0.043)	(0.045)	**(0.054)**	(0.057)
audiology	**0.787**	0.775	0.747	0.740
	(0.059)	(0.071)	(0.071)	(0.077)
bridges_version1	**0.603**	0.582	0.404	0.404
	(0.066)	(0.111)	(0.140)	(0.146)
car	**0.982**	0.931	0.955	0.894
	(0.014)	(0.023)	(0.030)	(0.023)
cylinder-bands	**0.686**	0.578	0.589	0.589
	(0.042)	(0.008)	(0.046)	(0.048)
glass	**0.704**	0.692	0.666	0.640
	(0.082)	(0.043)	(0.084)	(0.088)
iris	0.941	0.940	**0.957**	0.940
	(0.061)	(0.066)	**(0.042)**	(0.049)
kdd_synthetic_control	**0.922**	0.910	0.889	0.877
	(0.037)	(0.043)	(0.033)	(0.034)
segment	0.966	**0.970**	0.958	0.956
	(0.010)	**(0.010)**	(0.013)	(0.013)
semeion	0.953	**0.953**	0.935	0.933
	(0.021)	**(0.020)**	(0.018)	(0.018)
sick	0.987	0.987	0.987	**0.988**
	(0.003)	(0.003)	(0.006)	**(0.006)**
tep.fea	**0.650**	0.650	**0.650**	0.649
	(0.020)	(0.021)	**(0.020)**	(0.020)
vowel	0.821	**0.829**	0.821	0.698
	(0.040)	**(0.031)**	(0.040)	(0.041)
winequality-red	**0.619**	0.612	0.594	0.598
	(0.031)	(0.032)	(0.031)	(0.028)
winequality-white	0.605	**0.606**	0.562	0.562
	(0.025)	**(0.027)**	(0.017)	(0.021)

To evaluate the statistical significance of the accuracy results, we calculated the average Friedman rank for GE-J48, J48, GE-REPTree and REPTree: 1.66, 2.22, 2.72 and 3.41, respectively. The average rank suggests that GE-J48 outperforms the other methods regarding predictive accuracy. The calculation of Iman's F statistic resulted in $F_f = 7.46$. Critical value of $F(k-1,(k-1)(n-1)) = F(3,45)$ for $\alpha = 0.05$ is 2.81. Since $F_f > F_{0.05}(3,45)$ $(7.46 > 2.1)$, the null-hypothesis is rejected. We proceed with a post-hoc Nemenyi test to find which method provides better results in a pairwise fashion. The critical difference is $CD = 1.17$. The differences between the average rank of GE-J48 and the others are 0.56, 1.06 and 1.75, respectively. Then, we can confidently argue that the performance of using GE with J48 is significantly better than using the algorithm REPTree.

Figure 4 shows the Nemenyi's critical diagram, as suggested by Demsăr [28]. In this diagram, a horizontal line represents the axis on which we plot the average rank values of the methods. The axis is turned so that the lowest (best) ranks are to the left since we perceive the methods on the left side as better. When comparing all the criteria against each other, we connect the groups of criteria that are not significantly different through a bold horizontal line. We also show the critical difference given by the Nemenyi's test above the graph. We can see that GE-J48 is connected to J48 and GE-REPTree (no significant difference). GE-J48 is significantly better than REPTree (no line connecting them), whereas J48 and GE-REPTree are not.

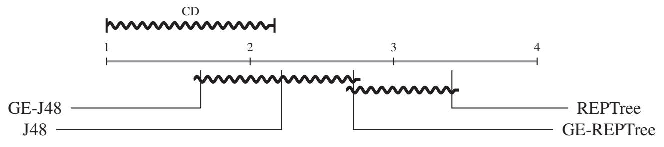

Fig. 4. Critical diagrams showing average ranks and Nemenyi's critical difference for Accuracy.

TABLE IV. F-MEASURE OF THE DECISION TREES GENERATED BY THE DT-INDUCTION ALGORITHMS FROM DATA SETS WITH (GE-J48 AND GE-REPTREE) OR WITHOUT (J48 AND REPTREE) THE NEW ATTRIBUTES. STANDARD DEVIATIONS ARE SHOWN BELOW THE F-MEASURE VALUES, BETWEEN BRACKETS.

data set	GE-J48	J48	GE-REPTree	REPTree
anneal	**0.983**	0.983	0.979	0.978
	(0.012)	(0.014)	(0.015)	(0.015)
arrhythmia	**0.644**	0.641	0.630	0.631
	(0.049)	(0.052)	(0.067)	(0.070)
audiology	**0.766**	0.754	0.713	0.704
	(0.069)	(0.082)	(0.077)	(0.085)
bridges_version1	**0.524**	0.523	0.290	0.290
	(0.070)	(0.109)	(0.102)	(0.106)
car	**0.982**	0.931	0.955	0.894
	(0.014)	(0.022)	(0.031)	(0.022)
cylinder-bands	**0.627**	0.423	0.494	0.494
	(0.075)	(0.009)	(0.083)	(0.087)
glass	**0.693**	0.677	0.635	0.615
	(0.083)	(0.051)	(0.090)	(0.097)
iris	0.940	0.939	**(0.957)**	0.939
	(0.063)	(0.068)	**(0.043)**	(0.050)
kdd_synthetic_control	**0.921**	0.909	0.888	0.875
	(0.038)	(0.044)	(0.033)	(0.035)
segment	0.966	**0.970**	0.958	0.956
	(0.010)	**(0.010)**	(0.013)	(0.013)
semeion	**0.952**	0.951	0.931	0.928
	(0.022)	(0.023)	(0.021)	(0.022)
sick	0.987	0.987	0.987	0.987
	(0.003)	(0.003)	(0.006)	(0.007)
tep.fea	0.610	0.610	0.610	0.610
	(0.022)	(0.023)	(0.022)	(0.022)
vowel	0.819	**0.827**	0.819	0.697
	(0.040)	**(0.030)**	(0.040)	(0.041)
winequality-red	**(0.614)**	0.605	0.576	0.580
	(0.029)	(0.032)	(0.035)	(0.030)
winequality-white	0.603	**0.605**	0.547	0.547
	(0.024)	**(0.025)**	(0.016)	(0.018)

Table IV shows the F-measure values for GE-J48, J48, GE-REPTree and REPTree. Again, one may observe that the best results are obtained by using the data set with the new attribute evolved by GE, either by using J48 or REPTree. If we consider the entire table, we can notice that GE-J48 or GE-REPTree present the best results in 11 out of 16 data sets. In two cases (sick and tep.fea), there was no difference among them, and in two cases (vowel and winequality-white) the additional attribute decreased the classifier's performance. However, it is important to observe the results for the data sets iris, segment, and, mainly, vowel. Even though the best performances were obtained without adding the new attribute by REPTree, J48 and J48, respectively, the use of GE increased the classifier's performance when adding the new attribute before applying another classifier, i.e., J48, REPTree and REPTree, respectively. In the case of vowel, the additional attribute increased the F-Measure in more than 10% (from 69.7% to 81.9%).

For the statistical analysis, the computed value of is $F_f = 12.59$. Since $F_f > F_{0.05}(3, 45)$ $(12.59 > 2.81)$, the null-hypothesis is rejected, and thus we can argue that there are significant differences among the four methods regarding F-measure. If we proceed with a post-hoc Nemenyi test, the critical difference is once again $CD = 1.17$, and we can observe that the differences between GE-J48 and REPTree (2.0) and between GE-J48 and GE-REPTree (1.38) are greater than CD (Figure 5). It is important to notice that GE-J48 once again achieves the lowest average rank, indicating it is the most suitable option in terms of F-Measure.

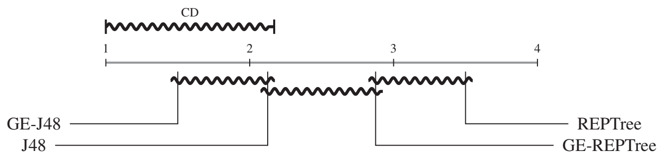

Fig. 5. Critical diagrams showing average ranks and Nemenyi's critical difference for F-Measure.

Table V shows the tree size values for GE-J48, J48, GE-REPTree and REPTree. Results show that GE-REPTree gerenrate smaller trees in most data sets, followed by REPTree without the previous use of the GE-based approach. J48 generates the smallest tree in only one case (cylinder_bands), when clearly generates only one leaf node representing the most frequent class, and that is why the results are not good in terms of accuracy and f-measure. GE-J48 generate the smallest trees in three cases: bridges_version1, tep.fea and vowel. Once again, it is possible to observe that the use of the GE-based approach also implies in smaller trees when compared to the same classifier but without the new attribute. This issue, however, was expected as the new attributes are composition of the original ones; once the attribute is used in a tree node (and it probably is, otherwise the results would be the same) it would be considered more representative and the tree would need less nodes (original attributes).

Regarding the statistical analysis, the computed value was $F_f = 6.58$. Since $F_f > F_{0.05}(3, 45)$ $(6.58 > 2.81)$, the null-hypothesis is rejected. Once again, we proceed with a post-hoc Nemenyi test, with $CD = 1.17$. According to Figure 6, we can easily observe that GE-REPTree and REPTree generate trees significantly smaller than the ones generated by both GE-J48 and J48. However, we can argue that this behaviour is because of the characteristic of REPTree algorithm, which usually generates smaller trees than J48 does. The most important here is to analyze if it is worth using GE to automatically generate a new attribute before inducing the final decision tree.

Since we have executed 5 independent runs over 10-

TABLE V. SIZE OF THE DECISION TREES GENERATED BY THE DT-INDUCTION ALGORITHMS FROM DATA SETS WITH (GE-J48 AND GE-REPTREE) OR WITHOUT (J48 AND REPTREE) THE NEW ATTRIBUTES. STANDARD DEVIATIONS ARE SHOWN BELOW THE TREE SIZE VALUES, BETWEEN BRACKETS.

data set	GE-J48	J48	GE-REPTree	REPTree
anneal	45.2 (8.1)	48.3 (6.5)	**39.3** **(4.5)**	40.4 (3.9)
arrhythmia	82.8 (5.5)	82.6 (5.8)	20.9 (6.3)	**20.0** **(6.0)**
audiology	48.1 (3.2)	50.4 (4.0)	**32.9** **(3.9)**	33.1 4.0
bridges_version1	**20.5** **(18.2)**	24.9 (20.7)	28.0 (27.3)	28.0 (28.5)
car	108.9 (21.5)	173.1 (6.5)	**107.4** **(23.9)**	137.3 (9.9)
cylinder-bands	4.2 (3.2)	**1.0** **(0.0)**	258.4 (212.3)	258.4 (221.5)
glass	43.2 (6.5)	44.8 (5.2)	20.2 (6.1)	**(18.8)** **(6.3)**
iris	7.5 (1.8)	8.0 (1.4)	**5.4** **(0.9)**	6.2 (1.7)
kdd_synthetic_control	34.0 (4.0)	37.8 (4.3)	**25.4** **(2.2)**	24.8 (3.0)
segment	79.3 (5.9)	80.6 (5.0)	52.6 (5.9)	**52.4** **(4.0)**
semeion	54.2 9.2	55.0 8.3	26.4 6.2	26.0 6.1
sick	46.8 (7.5)	46.9 (9.4)	**36.7** **(6.3)**	37.4 (6.9)
tep.fea	**6.7** **(1.0)**	8.2 (1.7)	7.6 (1.8)	9.2 (2.0)
vowel	**211.0** **(18.5)**	220.7 (20.7)	**211.0** **(18.5)**	258.7 (9.2)
winequality-red	391.2 (22.8)	387.0 (26.5)	120.9 (19.7)	**117.8** **(18.6)**
winequality-white	1356.1 (48.4)	1367.2 (58.4)	426.0 (36.8)	**412.8** **(51.0)**

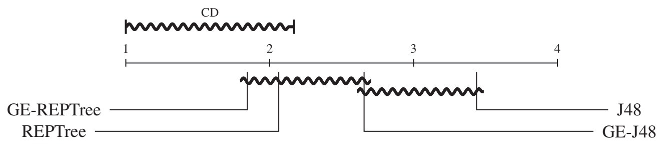

Fig. 6. Critical diagrams showing average ranks and Nemenyi's critical difference for Tree Size.

fold cross-validation, we evolved 50 new attributes for each approach, GE-J48 and GE-REPTree, for each data set. As examples, we present two of the new attributes. The first one (Equation 4) was evolved by GE-J48 over the data set "cylinder_bands". The second one (Equation 5) was generated by GE-REPTree over the data set "vowel". We have chosen these two data sets as examples because of the high contribution they provided to the classifiers's performance.

One may notice that the new attributes are small and that constant values are present, either for scaling or for shifting the values. As GE uses random sampling to generate these constants, it could be useful to employ a better method of optimizing them.

$$newAtt_{cylinder\_bands} = (\sqrt{\sqrt{4.68} * \sqrt{att_9}}) * att_0 \quad (4)$$

$$\begin{aligned} newAtt_{vowel} = 0.57 * (\sqrt{4.44} + \sqrt{4.67} \\ * (4.33 + (\sqrt{att_1} * (\sqrt{3.49} + att_3)))) \end{aligned} \quad (5)$$

VI. Conclusions and Future Work

In this work we investigated Grammatical Evolution (GE) for feature construction. GE is a flexible evolutionary algorithm that uses a context-free grammar and a string mapping to evolve computer programs; the resulting solution can then be mapped to the desired programming language. GE has been successfully applied in solving many problems and showed to be an adequate candidate method for the task. Here, the objective was to analyze whether useful features could be constructed using a small and simple grammar while related work employ bigger ones.

Our feature construction method was used for evolving a single feature to improve the predictive performance of two specific decision-tree induction algorithms on a wrapper approach. We tested the method on many well-known data sets and compared the performance using only the original features with that including the constructed feature.

As shown in the discussion, GE seems a good feature construction method. Both J48 and REPTree produced better classifiers (higher accuracy and F-measure values) and smaller trees for most data sets when using the evolved features. We believe that the results can be even better and thus many investigations can be done.

The main future work is to modify the grammar to generate more complex features to detect other kinds of relationships among the variables, for instance, other arithmetic and geometric functions. Also, we intend to test other classification algorithms and investigate a distinct feature construction method, such as the Genetic Programming, for comparison purposes.

Acknowledgements

The authors would like to thank Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) for funding this research.

References

[1] R. S. Michalski, *Pattern recognition as knowledge-guided computer induction*. Department of Computer Science, University of Illinois at Urbana-Champaign, 1978.

[2] V. V. de Melo and W. Banzhaf, “Kaizen programming for feature generation,” in *Genetic Programming Theory and Practice XIII*, ser. Genetic and Evolutionary Computation, R. Riolo, W. P. Worzel, and K. Groscurth, Eds. Ann Arbor, USA: Springer, May 2015, forthcomming.

[3] V. V. de Melo and W. Banzhaf, “Predicting high-performance concrete compressive strength using features constructed by kaizen programming,” in *2015 Brazilian Conference on Intelligent Systems, BRACIS 2015, Natal, Brazil, November 4-7, 2015*, 2015, pp. 80–85.

[4] V. V. de Melo, “Kaizen programming,” in *Genetic and Evolutionary Computation Conference, GECCO ’14, Vancouver, BC, Canada, July 12-16, 2014*, D. V. Arnold, Ed. ACM, 2014, pp. 895–902.

[5] D. Gavrilis, I. G. Tsoulos, and E. Dermatas, “Selecting and constructing features using grammatical evolution,” *Pattern Recognition Letters*, vol. 29, no. 9, pp. 1358–1365, 2008.

[6] D. Gavrilis, “Classification of fetal heart rate using grammatical evolution,” in *Signal Processing Systems Design and Implementation, 2005. IEEE Workshop on*. IEEE, 2005, pp. 425–429.

[7] G. M. Pagallo, “Adaptative decision tree algorithms for learning from examples (ph. d. thesis),” 1990.

[8] C. E. Brodley and P. E. Utgoff, *Multivariate versus univariate decision trees*. Citeseer, 1992.

[9] N. Indurkhya and S. M. Weiss, “Iterative rule induction methods,” *Applied Intelligence*, vol. 1, no. 1, pp. 43–54, 1991.

[10] H. Guo, L. B. Jack, and A. K. Nandi, “Feature generation using genetic programming with application to fault classification,” *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 35, no. 1, pp. 89–99, 2005.

[11] K. Krawiec, “Genetic programming-based construction of features for machine learning and knowledge discovery tasks,” *Genetic Programming and Evolvable Machines*, vol. 3, no. 4, pp. 329–343, 2002.

[12] K. Neshatian, M. Zhang, and M. Johnston, “Feature construction and dimension reduction using genetic programming,” in *AI 2007: Advances in Artificial Intelligence*. Springer, 2007, pp. 160–170.

[13] R. C. Barros, M. P. Basgalupp, A. C. P. L. F. de Carvalho, and A. A. Freitas, “Automatic Design of Decision-Tree Algorithms with Evolutionary Algorithms,” *Evolutionary Computation*, vol. 21, no. 4, 2013.

[14] R. Barros, M. Basgalupp, A. de Carvalho, and A. Freitas, “A survey of evolutionary algorithms for decision-tree induction,” *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 42, no. 3, pp. 291–312, May 2012.

[15] R. C. Barros, A. T. Winck, K. S. Machado, M. P. Basgalupp, A. C. P. L. F. de Carvalho, D. D. Ruiz, and O. S. de Souza, “Automatic design of decision-tree induction algorithms tailored to flexible-receptor docking data,” *BMC Bioinformatics*, vol. 13, 2012.

[16] R. C. Barros, M. P. Basgalupp, A. A. Freitas, and A. C. P. L. F. de Carvalho, “Evolutionary Design of Decision-Tree Algorithms Tailored to Microarray Gene Expression Data Sets,” *IEEE Transactions on Evolutionary Computation*, vol. in press, 2014.

[17] C. Ryan and M. O’Neill, “Grammatical evolution: A steady state approach,” *Late Breaking Papers, Genetic Programming*, vol. 1998, pp. 180–185, 1998.

[18] M. O’Neill and C. Ryan, “Grammatical evolution,” *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 349–358, August 2001.

[19] J. R. Quinlan, *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann, 1993.

[20] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The weka data mining software: an update,” *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[21] K. Bache and M. Lichman, “UCI machine learning repository,” 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[22] Z. Zheng, “Constructing nominal x-of-n attributes,” in *IJCAI*, vol. 95. Citeseer, 1995, pp. 1064–1070.

[23] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, MA, USA: MIT Press, 1992.

[24] S. Ahmed, M. Zhang, L. Peng, and B. Xue, “Multiple feature construction for effective biomarker identification and classification using genetic programming,” in *Proceedings of the 2014 conference on Genetic and evolutionary computation*. ACM, 2014, pp. 249–256.

[25] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Addison-Wesley, 2005.

[26] M. P. Basgalupp, R. C. Barros, A. C. de Carvalho, and A. A. Freitas, “Evolving decision trees with beam search-based initialization and lexicographic multi-objective evaluation,” *Information Sciences*, vol. 258, pp. 160 – 181, 2014.

[27] S. Luke, L. Panait, G. Balan, S. Paus, Z. Skolicki, R. Kicinger, E. Popovici, K. Sullivan, J. Harrison, J. Bassett, R. Hubley, A. Desai, A. Chircop, J. Compton, W. Haddon, S. Donnelly, B. Jamil, J. Zelibor, E. Kangas, F. Abidi, H. Mooers, J. O’Beirne, K. A. Talukder, S. McKay, and J. McDermott, “ECJ - A Java-based Evolutionary Computation Research System,” https://cs.gmu.edu/∼eclab/projects/ecj/.

[28] J. Demšar, “Statistical comparisons of classifiers over multiple data sets,” *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, 2006.

[29] R. Iman and J. Davenport, “Approximations of the critical region of the friedman statistic,” *Communications in Statistics*, pp. 571–595, 1980.