Evolution of Scikit-Learn Pipelines with Dynamic Structured Grammatical Evolution

Filipe Assunção[1,2(✉)], Nuno Lourenço[1], Bernardete Ribeiro[1], and Penousal Machado[1]

[1] CISUC, Department of Informatics Engineering, University of Coimbra, Coimbra, Portugal
{fga,naml,bribeiro,machado}@dei.uc.pt
[2] LASIGE, Department of Informatics, Faculdade de Ciencias, Universidade de Lisboa, Lisbon, Portugal

Abstract. The deployment of Machine Learning (ML) models is a difficult and time-consuming job that comprises a series of sequential and correlated tasks that go from the data pre-processing, and the design and extraction of features, to the choice of the ML algorithm and its parameterisation. The task is even more challenging considering that the design of features is in many cases problem specific, and thus requires domain-expertise. To overcome these limitations Automated Machine Learning (AutoML) methods seek to automate, with few or no human-intervention, the design of pipelines, i.e., automate the selection of the sequence of methods that have to be applied to the raw data. These methods have the potential to enable non-expert users to use ML, and provide expert users with solutions that they would unlikely consider. In particular, this paper describes AutoML-DSGE – a novel grammar-based framework that adapts Dynamic Structured Grammatical Evolution (DSGE) to the evolution of Scikit-Learn classification pipelines. The experimental results include comparing AutoML-DSGE to another grammar-based AutoML framework, Resilient Classification Pipeline Evolution (RECIPE), and show that the average performance of the classification pipelines generated by AutoML-DSGE is always superior to the average performance of RECIPE; the differences are statistically significant in 3 out of the 10 used datasets.

Keywords: Automated Machine Learning · Scikit-Learn · Dynamic Structured Grammatical Evolution

1 Introduction

Nowadays, with the ever-growing amount of collected information the challenge is not concerned with the lack of information, but rather on how to design efficient Machine Learning (ML) models that can extract useful knowledge, or

P. A. Castillo et al. (Eds.): EvoApplications 2020, LNCS 12104, pp. 530–545, 2020.
https://doi.org/10.1007/978-3-030-43722-0_34

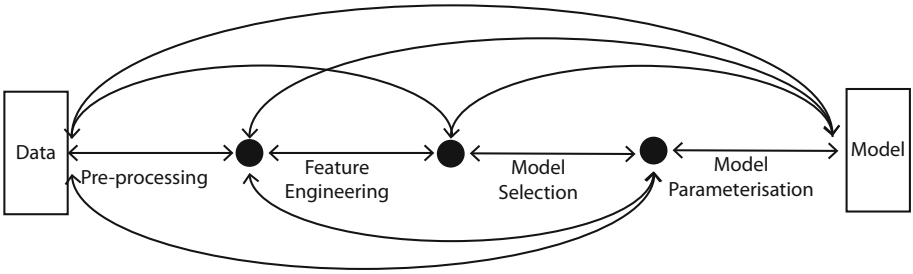

Fig. 1. From data to ML model deployment.

aid in the automation of daily-life tasks. Typically, to deploy a Machine Learning (ML) system we need to follow a pre-defined number of steps: (i) pre-process the data; (ii) design, extract, and select features, i.e., data characteristics; (iii) select the most appropriate ML model; and (iv) parameterise the ML model. The flow of the iterative steps that one must traverse from the data to the model is depicted in Fig. 1: the multiple steps are all interconnected, which means that for example in case we have already a model, but we acknowledge that the set of features is not the most adequate one we may be thrown back to the beginning of the process again. In addition, even when the practitioner is a ML expert, and is well aware of which models are more adequate for particular tasks, it still needs to design features, which are domain-dependent, and therefore often require domain-expertise, and sometimes multidisciplinary teams.

To overcome the difficulty caused by the correlation between the multiple choices that have to be made prior to deploying a ML system we can resort to Automated Machine Learning (AutoML). In brief words Automated Machine Learning (AutoML) concerns searching for the most effective ML models for a particular task. One of the key-advantages of AutoML is that it does not require human input, and consequently the gain is twofold: (i) on the one hand it empowers non-expert users with the ability to apply ML models to their problems; (ii) on the other hand, it opens the door to novel solutions, that a human-expert would potentially neglect.

The current work focuses on AutoML applied to classification datasets. The common approach of AutoML to this sort of problems is to evolve a classification pipeline, i.e., an ordered sequence of tasks that are performed to accurately distinguish between the different classes of the problem. The pipeline tasks can be any known form of data pre-processing; feature design, extraction, or selection; or ML algorithm. In particular, we evolve Scikit-Learn [1] pipelines with Dynamic Structured Grammatical Evolution (DSGE) [2]. Our main contributions are:

- The proposal of a new grammar-based AutoML framework based on Dynamic Structured Grammatical Evolution (DSGE): AutoML-DSGE;
- The release of the framework as open-source, available on GitHub: https://github.com/fillassuncao/automl-dsge;

- The performance of a wide set of experiments on multiple classification tasks;
- The comparison of AutoML-DSGE to previous AutoML methods. The results show that the results of AutoML-DSGE are always superior to those reported by other grammar-based AutoML methods, and are statistically superior in 3 out of the 10 used datasets.

The remainder of the paper is structured as follows. Section 2 surveys multiple AutoML methods; Sect. 3 describes DSGE; Sect. 4 details the evolution of Scikit-Learn classification pipelines with DSGE; Sect. 5 analyses the experimental results; and Sect. 6 draws conclusions and addresses future work.

2 Related Work

The most common and widely used form of AutoML is grid search: the best parameterisation of a ML model is discovered by an exhaustive search of all the combinations of a grid of parameters. However, grid search suffers from the curse of dimensionality, i.e., the explosion in the number of parameters drastically increases the amount of setups that need to be tested. To deal with the previous we can instead use grid search methods that seek to narrow the number of setups, for example by adapting the resolution of the grid in run-time [3]. Nonetheless, grid search has the advantage that it is highly parallelisable. To overcome the issue of having to explore the entire grid of hyper-parameters we may instead apply random search. While grid search performs an exhaustive enumeration of the domain, random search selects the combinations of the hyper-parameters in a stochastic manner. Random search is as parallelisable as grid search. Nonetheless, it is non-adaptive [4], and with very high dimensional search spaces it also struggles to find near-optimal solutions. According to Bergstra et al., given the same computational time, random search is able to discover better parameterisations for Artificial Neural Networks (ANNs) than grid search [5,6].

An alternative to grid and random search are Bayesian methods [7], which model probabilistically the behaviour of the system, in order to drive search towards regions of the domain that are prone to generate good parameterisations. Snoek et al. applied Bayesian optimisation to tune the parameters of the Branin-Hoo function, Logistic Regression, Online Linear Discriminant Analysis, Latent Structured Support Vector Machines, and Convolutional Neural Networks [8]. Bergstra et al. have demonstrated that statistical methods can perform better at hyper-parameter optimisation [9] than manual tuning or random search. Other class of heuristic approach is Evolutionary Computation (EC), which has also been widely used to optimise ML algorithms (e.g., [10,11]).

The majority of the methods mentioned until now focus on the optimisation of a specific ML model. Nonetheless, the ultimate goal of AutoML is to fully automate the entire process: from the data pre-processing, and feature design and selection up to the model choice and parameterisation. Recently, there have been competitions seeking to promote such systems; an example is ChaLearn [12]. The challenge is organised into 6 increasingly difficult levels (preparation, novice,

intermediate, advanced, expert, and master), where the ultimate goal is to "create the perfect black box eliminating the human in the loop" [13].

Weka [14] and Scikit-learn [1] are examples of two ML libraries that enable users to explore their data and easily deploy learning models. They make available stable implementations of the vast majority of ML methods, but despite providing default parameterisation they are not suit for effectively solving all problems. Auto-WEKA [15,16], Tree-based Pipeline Optimization Tool (TPOT) [17], Hyperopt-Sklearn [18], Auto-Sklearn [19], and Resilient Classification Pipeline Evolution (RECIPE) [20], are examples of methods that aim at evolving the pipelines for the Weka and Scikit-learn libraries, from the pre-processing of the raw data to the parameterisation of the model to be used (in essence they automate the flow-chart depicted in Fig. 1). Except for TPOT and Resilient Classification Pipeline Evolution (RECIPE), all the previous methodologies are based on Bayesian optimisation; TPOT and RECIPE use Genetic Programming (GP). The goal is to search for Weka or Scikit-Learn pipelines, i.e., sequences of the libraries' primitives that perform feature selection and classification. The frameworks are not only responsible for selecting the primitives but also promote their parameterisation. Auto-Weka, Hyperopt-Sklearn, Auto-Sklearn and RECIPE generate pipelines of fixed size; TPOT allows the generation of pipelines of unrestricted size, i.e., it does not have a fixed number of pre-processors, and multiple copies of the dataset can be used in simultaneous, so that multiple methods are applied to it, and then the features combined. Whilst the majority of these approaches target the maximisation of the classification performance, in addition TPOT also seeks for compact pipelines.

The focus of the current work is on AutoML approaches based on EC. In particular, we are interested in grammar-based methods, such as RECIPE. The main advantage of grammar-based methods over others is that they facilitate the definition of the search space, and thus in case we have a-priori knowledge about the problem we can bias the grammar. On the other hand, the grammar enables the framework to be easily extended: to add more methods to the search space we just require the definition of new production rules. To the best of our knowledge, RECIPE is the only grammar-based AutoML framework that aims at optimising classification pipelines. The current paper introduces AutoML-DSGE and compares it to RECIPE. AutoML-DSGE is based on DSGE, which is detailed next.

3 Dynamic Structured Grammatical Evolution

To properly introduce DSGE [2] we must start by detailing Structured Grammatical Evolution (SGE) [21], which is a variant of Grammatical Evolution (GE) [22]. The three methods are grammar-based GP approaches, and thus the search space is defined by means of a Context-Free Grammar (CFG). CFGs are rewriting systems, and thus the grammar, G, can be formally defined by a 4-tuple $G = (N, T, P, S)$, where: (i) N is the set of non-terminal symbols; (ii) T is the set of terminal symbols; (iii) P is the set of production rules of the form

$x ::= y$, $x \in N$ and $y \in \{N \cup T\}^*$; and (iv) S is the start symbol (or axiom). An example of a CFG is shown in Fig. 2. The main difference between the methods lies on the encoding of the individuals, and thereby on the genotype decoding procedure.

The individuals in GE are encoded as linear ordered sequences of integers; each integer represents a derivation step and is called a codon. The genotype to phenotype mapping works by reading the codons sequentially, from left to right. Starting from the axiom the mapping procedure iteratively decides which production rule should be applied to expand the leftmost non-terminal symbol. To select the production rule the modulo mathematical operation (%) is used to find the remainder after the division of the codon by the number of possibilities for expanding the leftmost non-terminal symbol. The remainder defines the expansion possibility that should be applied to the leftmost non-terminal symbol. No codon is read when there is only a possibility for expanding a non-terminal symbol. On the other hand, grammars can be recursive, and thus the number of codons may be insufficient; in such cases the sequence of codons is re-used from the start (wrapping). To avoid entering an infinite wrapping loop, or generating solutions that are too complex to be evaluated, a maximum number of wrappings is set, and when this bound is reached the mapping procedure is halted, and the individual is assigned the worst possible fitness value.

The drawbacks commonly pointed to GE are low locality and high redundancy [23,24]. The locality measures how the changes in the genotype impact the phenotype. In GE there is not a one-to-one mapping between the codons and the non-terminal symbols, and therefore it is easy for a change in one of the codons to affect all the derivation steps from that point on-wards (low locality). On the other hand, the redundancy is concerned to the fact that in GE it is possible that different genotypes generate the same phenotype because of the modulo operation used on the decoding procedure.

SGE solves the limitations of GE by introducing a new genotypic representation that defines a one-to-one mapping between the codons and the non-terminal symbols, i.e., instead of a single ordered sequence of codons the genotype is composed by multiple independent ordered sequences of codons, one for each non-terminal symbol. The size of each sequence of codons is of the maximum number of possible expansions for the non-terminal symbol it encodes, and thus there is no wrapping. The use of the modulo operation is not required as we know exactly which non-terminal symbol the codon encodes.

In SGE the genotypes encode more codons than the ones used in the decoding procedure, and consequently the genetic operators may easily act upon non-coding genes. This under some circumstances can slow down evolution. To prevent this effect, the genotype of DSGE is similar to those of SGE with one main difference: it only encodes the codons strictly required for decoding the individual. In case mutations affect the amount of necessary codons, the genotype is expanded. In this paper we use DSGE; the code for DSGE can be found in the GitHub repository https://github.com/nunolourenco/sge3.

Table 1. Scikit-Learn classes that are allowed to be part of the pipelines.

Pre-processing	Feature manipulation	Classification
Imputer	VarianceThreshold	ExtraTreeClassifier
Normalizer	SelectPercentile	DecisionTreeClassifier
MinMaxScaler	SelectFpr	GaussianNB
MaxAbsScaler	SelectFwe	BernouliNB
RobustScaler	SelectFdr	MultinominalNB
StandardScaler	RFE	SVC
	REFCV	NuSVC
	SelectFromModel	KNeighborsClassifier
	IncrementalPCA	RadiusNeighborsClassifier
	PCA	NearestCentroid
	FastICA	LDA
	GaussianRandomProjection	QDA
	SparseRandomProjection	LogisticRegression
	RBFSampler	LogisticRegressionCV
	Nystroem	PassiveAggressiveClassifier
	FeatureAgglomeration	Perceptron
	PolynomialFeatures	Ridge
		RidgeCV
		AdaBoostClassifier
		GradientBoostingClassifier
		RandomForestClassifier
		ExtraTreesClassifier

4 AutoML-DSGE

The goal of this paper is to introduce a new framework, to which we call AutoML-DSGE, that adapts DSGE to the evolution of classification pipelines. In particular, we optimise Scikit-Learn [1] pipelines. Next, we define pipelines (Sect. 4.1), the used grammar (Sect. 4.2), and detail the evolution of pipelines using DSGE (Sect. 4.3). The code for AutoML-DSGE is released as open-source software, and can be found in the GitHub repository https://github.com/fillassuncao/automl-dsge.

4.1 Pipelines

In the field of ML a classification pipeline is defined as an ordered set of operations that are performed to the data instances in order to accurately separate them in the multiple classes of the dataset. The operations in the pipeline can be grouped into 3 disjoint sets: (i) data pre-processing; (ii) feature design and

```
<pipeline> ::= <preprocessing> <algorithm>                                  (1)
             | <algorithm>                                                  (2)
<preprocessing> ::= <imputation> | <bounding> | <dimensionality> |          (3)
             | <binarizer> | <imputation> <bounding>                       (4)
             | <imputation> <binarizer>                                     (5)
             | ...                                                          (6)
<imputation> ::= preprocessing:imputer <strategy_imp>                       (7)
<strategy_imp> ::= strategy:mean | strategy:median | strategy:most_frequent (8)
             ...                                                            (9)
             ...                                                           (10)
<algorithm> ::= <strong> | <weak> | <tree_ensemble>                        (11)
             ...                                                           (12)
             ...                                                           (13)
<weak> ::= <nearest> | <discriminant> | ...                                (14)
             ...                                                           (15)
             ...                                                           (16)
<nearest> ::= classifier:radius_neighbors <radius> <weights>              (17)
             <k_algorithm> <leaf_size> <p> <d_metric>                      (18)
<radius> ::= radius:RANDFLOAT(1.0,30.0)                                    (19)
<weights> ::= weights:uniform | weights:distance                           (20)
<k_algorithm> ::= algorithm:auto | algorithm:brute| ...                    (21)
<leaf_size> ::= leaf_size:RANDINT(5,100)                                   (22)
             ...                                                           (23)
             ...                                                           (24)
```

Fig. 2. CFG used by AutoML-DSGE for optimising Scikit-Learn pipelines.

selection; and (iii) classification. Table 1 enumerates the methods that are considered to form the pipelines in the current work. Recall that we focus on classification pipelines, and thus only classification algorithms are taken into account. Nonetheless, the extension of the approach to regression algorithms is straightforward. We will optimise Scikit-Learn pipelines, and thus the methods in the table are Scikit-Learn implementations. Further details can be found in https://scikit-learn.org/stable/user_guide.html.

4.2 Grammar

The grammar used by AutoML-DSGE describes the search space of the Scikit-Learn classification pipelines. The grammar is shown in Fig. 2. The production rules are only partially shown because of space constraints: the grammar is comprised of 89 production rules that encode the different pipeline methods and

their parameterisation. The complete grammars can be found in https://github.com/fillassuncao/automl-dsge/tree/master/sge/grammars. There is a separate grammar for each dataset because of specific dataset parameters, e.g., number of features. The used grammars are adapted from the grammars used by RECIPE, which is the method we compare AutoML-DSGE to.

The axiom of the grammar is the pipeline non-terminal symbol, and consequently the pipeline can be found by either pre-processing and classification methods (line 1) or just by the classification method (line 2). The current version of AutoML-DSGE does not consider ensembles. The extension of AutoML-DSGE to enable the optimisation of ensembles could be easily introduced by adding a recursive production rule to build pipelines with more than one classifier algorithm, each a voter of the ensemble. The pre-processing methods manipulate the dataset and features (lines 3–6), and the classification methods cover a wide range of ML approaches, amongst which, are clustering methods, Support Vector Machines (SVMs), trees, or ANNs (lines 11–18). In more detail, the pipeline methods are encoded as follows: the pre-processing and classification methods are encoded respectively by the preprocessing and classifier tags, that are placed before the method name (e.g., classifier:radius_neighbors in line 17). The method name must match the name of the function that is used in the mapping from the phenotype to the Scikit-Learn interpretable code (see Sect. 4.3). The same rationale is applied to the method parameters, where the parameter name precedes the parameter value. The parameters can be of three types: (i) closed choice, e.g., the weights parameter, in line 20, that can assume the values uniform or distance; (ii) random integer, e.g., the leaf size parameter in line 22; or (iii) random float, e.g., the radius parameter in line 19.

The search space of AutoML-DSGE, i.e., the number of possible combinations of the grammar is greater than 9.39×10^{17}. The continuous parameters can generate an infinite number of possibilities, and thus are not considered in the search space size. In addition, the parameters related to the number of features are also not taken into account because they are problem dependent.

4.3 Evolution of Pipelines

The pipelines are evolved using DSGE, and therefore, a population of individuals is continuously evolved throughout a given number of generations, until a stop criteria is met. Each individual encodes a different pipeline. The core of the representation of the individuals in AutoML-DSGE is similar to the representation scheme used in DSGE, with one main difference related to the need to directly keep real values in the genotype. Otherwise, they would have to be encoded by production rules, such as:

Table 2. Description of the used datasets.

Dataset	#Inst.	#Feat.	Feat. types	#Classes	Missing
Breast Cancer	699	9	Integer	2	Yes
Car Evaluation	1728	5	Categorical	4	No
Caenorhabditis Elegans	478	765	Binary	2	No
Chen-2002	179	85	Real	2	No
Chowdary-2006	104	182	Real	2	No
Credit-G	1000	20	Real/Categorical	2	No
Drosophila Melanogaster	119	182	Real	2	No
DNA-No-PPI-T11	135	104	Real/Categorical	2	Yes
Glass	214	9	Real	7	No
Wine Quality-Red	1599	11	Real	10	No

<randfloat> ::= <signal> <rec-number> .<rec-number>
<signal> ::= − | +
<rec-number> ::= <number> | <number> <number-recursive>
<number> ::= 0 | 1 | 2 | 3 | 4
5 | 6 | 7 | 8 | 9

The encoding of real values by means of production rules has two main disadvantages. On the one hand it enlarges the search space. On the other hand there is no easy way to control the limits (minimum and maximum) of the generated real values. In case the search space encompasses two or more real values with different ranges there would be the need for different production rules, one for each real value range. Because of the aforementioned we encode the integers and floats directly, as real values. When expanding the grammar, when we reach a terminal symbol that is either RANDINT or RANDFLOAT we store a tuple in the genotype. The tuple has the format (rand-type, rand-min, rand-max, rand-value), where rand-type can assume integer of float, the rand-min and rand-max are the minimum and maximum limits of the range, and the rand-value is the randomly generated value of the type rand-type, and within the [rand-min, rand-max] range. The tuple is necessary for performing the mutation, i.e., when a mutation is applied to an individual and it is required to generate a new random value for a specific parameter we must know its type and allowed range.

DSGE is a grammar-based approach, and thus the genotype is completely separate from the phenotype. The phenotype does not directly represent a trainable pipeline. Consequently, for assessing the fitness of the individuals we have to perform two sequential steps: (i) map the genotype to the phenotype; (ii) map the phenotype to Scikit-Learn interpretable model. To map the genotype to the phenotype the decoding procedure of DSGE is adapted: the only difference lies in the decoding of the real-values, where the value in the last position of the tuple is read. The phenotype of AutoML-DSGE is readable, despite not

Table 3. Experimental parameters.

Parameter	Value
Number of runs	30
Number of generations	100
Population size	100
Mutation rate	10%
Crossover rate	90%
Elitism	5 individuals
Tournament size	2
Max. pipeline train time	5 min
Max. #generations without improvement	5

being Scikit-Learn executable code. The readability of the phenotype is facilitated by the fact that each parameter has the parameter name associated to the value; an example of a phenotype is "classifier:random_forest criterion:gini max_depth:None n_estimators:50 min_weight_fraction_leaf:0.01 ...".

To map the phenotype to a Scikit-Learn interpretable pipeline we have to traverse the phenotype linearly from left to right and for each pre-processing or classifier method create the corresponding Scikit-Learn object. Therefore, for each method in the grammar we have to build a function that creates the Scikit-Learn object: the function receives all the parameters that are encoded in the grammar and outputs the Scikit-Learn object. Whenever the grammar is extended to include more methods we have to create the corresponding functions.

To evaluate the evolved pipelines we use cross-validation (with 3 folds). In the current paper the fitness is the average of the performances on the cross-validation. The metric used to evaluate the performance is the F-measure. We decide for this metric because some of the datasets where we will be conducting the experiments are highly unbalanced.

The goal of AutoML is to generate (automatically) effective Scikit-Learn classification pipelines that non-expert ML users can deploy in their problems and domains. With this in mind, similarly to other approaches, we limit the train time of each pipeline to a maximum CPU time, that in this paper is set to five minutes. For the same reason evolution is halted when there is no improvement for five generations.

5 Experimentation

To investigate the ability of AutoML-DSGE to generate effective classification Scikit-Learn pipelines we apply it to the classification of 10 datasets, which are described in Sect. 5.1. The experimental setup is detailed in Sect. 5.2, and the analysis of the evolutionary results, and comparison to the pipelines generated by RECIPE is carried out in Sect. 5.3.

Table 4. AutoML-DSGE, and RECIPE comparative performance. The results are averages of 30 independent runs.

Dataset	AutoML-DSGE	RECIPE	p-value
Breast Cancer	**0.9568 ± 0.0296**	0.9311 ± 0.0798	**0.0264** (++)
Car Evaluation	**0.9964 ± 0.0068**	0.9962 ± 0.0079	0.9761
Caenorhabditis Elegans	**0.6140 ± 0.0644**	0.6049 ± 0.0681	0.7948
Chen-2002	**0.9451 ± 0.0413**	0.9292 ± 0.0618	0.3371
Chowdary-2006	**0.9970 ± 0.0163**	0.9812 ± 0.0514	0.0679
Credit-G	**0.7400 ± 0.0370**	0.7075 ± 0.0359	**0.0008** (+++)
Drosophila Melanogaster	**0.6679 ± 0.1001**	0.6353 ± 0.1518	0.2585
DNA-No-PPI-T11	**0.7114 ± 0.1194**	0.7021 ± 0.0761	0.9681
Glass	**0.7628 ± 0.1095**	0.7325 ± 0.1021	0.0524
Wine Quality-Red	**0.6600 ± 0.0387**	0.6430 ± 0.0422	**0.0257** (++)

5.1 Datasets

To enable a fair comparison between AutoML-DSGE and RECIPE we conduct the experiments on the same datasets used by RECIPE: 10 datasets – 5 from the University of California Irvine (UCI) ML repository [25], and 5 from bio-informatics [26–28]. A summary of the dataset characteristics is shown in Table 2. The table provides information on the number of instances (#Inst.), number of features (#Feat.), type of features (Feat. types), number of classes (#Classes), and if there are or not missing values in the dataset (Missing).

5.2 Experimental Setup

The parameters required for performing the experiments contained in this article are described in Table 3. The parameters are the same for AutoML-DSGE and RECIPE. The maximum CPU train time is measure in minutes, and thus it is important to mention that the experiments are performed in a dedicated server with an Intel(R) Core(TM) i7-5930K CPU @ 3.50 GHz, and 32 GB of RAM.

The code used for AutoML-DSGE, and RECIPE can be found, respectively, in the GitHub repositories github.com/laic-ufmg/Recipe/, and github.com/fillassuncao/automl-dsge. The code of RECIPE was modified to include the evolution stop criteria based on a maximum number of generations without improvement, which despite described in the framework paper [20], is not included in the current code version.

To enable the comparison of results we apply the same dataset partitioning scheme used in RECIPE: all datasets are split using a 10-fold cross-validation strategy; and thus as we perform 30 evolutionary runs each fold is kept as the test set three times, and the remaining used for training the pipelines. During each run, the test set is kept aside from evolution, and the train set is used to train the pipelines with cross-validation (3 folds). By the end of evolution, the

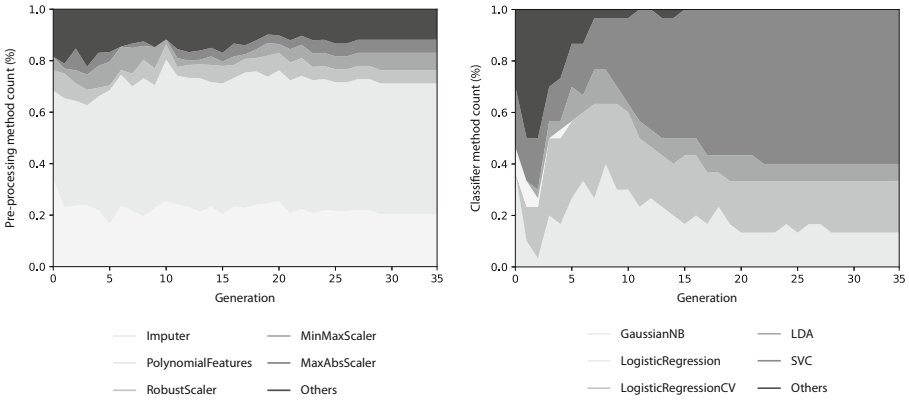

Fig. 3. Stacked area charts of the AutoML-DSGE evolution of the pre-processing (left) and classification (right) methods on the Car dataset. The results reflect the percentage of the best pipelines that use each of the methods.

best pipeline is trained using all the train data and applied to the test set. The evolution is conduced using the grammar of Fig. 2.

To establish the pair-wise comparison of the results, and check whether or not the differences between AutoML-DSGE and RECIPE are statistically significant we use the Wilcoxon Signed-Rank test, with a significance level of 5%. Further, for the statistically significant differences we compute the effect size.

5.3 Experimental Results

To compare the pipelines generated by AutoML-DSGE and RECIPE we conduct evolution for the same datasets, and using equivalent grammatical formulations, i.e., the search space is the same for both frameworks. The test performance (f-measure), for each dataset is presented in Table 4. The results are averages of 30 independent runs. A f-measure marked in bold indicates the approach that reports the highest average performance. In addition, the table also reports the p-values for the pair-wise comparisons between the two approaches, and bold p-values indicate statistically significant differences. The effect-size is denoted in brackets after the p-value, with +, ++, and +++ denoting small ($0.1 \leq r < 0.3$), medium ($0.3 \leq r < 0.5$), and large ($r \geq 0.5$) effect sizes, respectively.

The analysis of the results indicates that AutoML-DSGE reports results that are always superior to those obtained by RECIPE. In addition to the higher average, the standard deviation is lower in the AutoML-DSGE results in 7 out of 10 datasets, i.e., for the considered datasets AutoML-DSGE generates more consistently higher results. These differences are statistically significant in 3 datasets (Breast Cancer, Credit-G, and Wine Quality-Red). The effect size is medium twice, and high once. AutoML-DSGE is never worse than RECIPE.

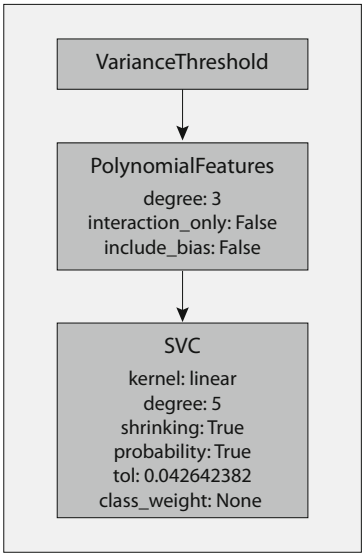

Fig. 4. Best pipeline generated for classifying the Car dataset. Each box represents a pipeline method and its parameterisation.

The results of Table 4 report the average performance of the 30 evolutionary runs, for each dataset. Nonetheless, as we are optimising ML methods we investigate the generalisation ability of the generated pipelines. To this end, we compute the average difference between the evolutionary, and test performances for the 10 datasets. Except for the Chowdary-2006 and Car datasets, the average difference between the evolutionary and test performance is lower in AutoML-DSGE than in RECIPE. Considering all datasets, the average difference between the evolutionary and test set performance is of approximately 0.0328 in AutoML-DSGE and of 0.0589 in RECIPE. This proves that the tendency to overfit is lower in AutoML-DSGE, as it reports more often than RECIPE evolutionary performances that are closer to the test ones.

To analyse the structure of the pipelines evolved by AutoML-DSGE we inspect the methods that compose them. Due to space constraints we focus on the Car dataset, as it is the dataset where, on average, more generations are performed. Figure 3 shows the evolution of the pre-processing and classification methods of the best individuals as generations proceed. The results show the evolution of the percentage of the runs that use each of the pre-processing and classification methods. Recall that the different evolutionary runs can differ in the number of performed generations, and therefore to avoid a misleading representation of the evolution of the methods that compose the pipelines we consider that all runs have the same number of generations. That is, we consider that all runs evolve for the same number of generations as the longer run (in this case 35 generations). For the evolutionary runs that perform less generations we keep the last generation (which is the best found solution) for the remainder of

the generations. The results show that, for the Car dataset, the pre-processing methods distribution does not change as evolution proceeds. On the other hand, a different behaviour is noticeable on the classifier methods, that converge to the SVC, and LogisticRegression (or LogisticRegressionCV) method. The evolution also shows that evolution is focused on the methods that are more effective for that specific dataset. Otherwise, the used methods would be more diverse, and the percentage of the Others would be higher. In particular, we plot in Fig. 4 the best pipeline found for classifying the Car dataset. We also inspect the evolutionary patterns in the remaining datasets and acknowledge similar conclusions. It is however important to point out that for the different datasets evolution focuses on different pre-processing and classification methods.

Ultimately, AutoML-DSGE generates no invalid pipelines. After investigating the pipelines that were assigned with the worse possible fitness we conclude that their train is halted because they are unable to train in the maximum granted CPU time of five minutes, or because they run out of memory.

6 Conclusions and Future Work

Prior to the deployment of a ML model there are a number of choices that have to be made. There is the need to pre-process the dataset, design, extract and select features, and decide which ML model is the most adequate. On top of that, all this sequential choices are correlated, meaning that one affects multiple others. The choices that have to be made require both domain-specific, and ML expertise. In an effort to facilitate the widespread use of ML models we introduce a novel AutoML framework: AutoML-DSGE.

AutoML-DSGE is a grammar-based AutoML approach, and thus the search space is defined in a human-readable CFG. This key-point of the framework enables the easy adaptation of AutoML-DSGE to tackle different problems using a wide set of methods. Further, it eases the introduction of a-priori knowledge in the search and tuning of the pipelines. The current version of the framework focuses on the optimisation of Scikit-Learn classification pipelines. The code is released as open-source software, and can be found in the GitHub repository: https://github.com/fillassuncao/automl-dsge.

We compare the performance AutoML-DSGE to RECIPE, which to the best of our knowledge is the only grammar-based AutoML framework. The methods are compared on 10 datasets from different domains. The results show that the pipelines generated by AutoML-DSGE surpass in performance the ones obtained by RECIPE; the average performances of AutoML-DSGE are always superior to RECIPE, and are statistically superior in 3 datasets (with medium and large effect sizes). Moreover, AutoML-DSGE is less prone to overfitting than RECIPE.

Future work will be divided into 4 independent research lines: (i) apply AutoML-DSGE to a wider set of benchmarks; (ii) extend the framework to regression problems; (iii) introduce ensembling and stacking methods; and (iv)

enable the user to select between the Weka and Scikit-Learn ML libraries, or even own implemented methods.

Acknowledgments. This work is partially funded by: Fundação para a Ciência e Tecnologia (FCT), Portugal, under the PhD grant agreement SFRH/BD/114865/2016, the project grant DSAIPA/DS/0022/2018 (GADgET), and is based upon work from COST Action CA15140: ImAppNIO, supported by COST (European Cooperation in Science and Technology): www.cost.eu. We also thank the NVIDIA Corporation for the hardware granted to this research.

References

1. Pedregosa, F., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
2. Lourenço, N., Assunção, F., Pereira, F.B., Costa, E., Machado, P.: Structured grammatical evolution: a dynamic approach. In: Ryan, C., O'Neill, M., Collins, J.J. (eds.) Handbook of Grammatical Evolution, pp. 137–161. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78717-6_6
3. Jiménez, Á.B., Lázaro, J.L., Dorronsoro, J.R.: Finding optimal model parameters by deterministic and annealed focused grid search. Neurocomputing **72**(13–15), 2824–2832 (2009)
4. Young, S.R., Rose, D.C., Karnowski, T.P., Lim, S., Patton, R.M.: Optimizing deep learning hyper-parameters through an evolutionary algorithm. In: MLHPC@SC, pp. 4:1–4:5. ACM (2015)
5. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: NIPS, pp. 2546–2554 (2011)
6. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**, 281–305 (2012)
7. Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., de Freitas, N.: Taking the human out of the loop: a review of Bayesian optimization. Proc. IEEE **104**(1), 148–175 (2016)
8. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. In: NIPS, pp. 2960–2968 (2012)
9. Bergstra, J., Yamins, D., Cox, D.D.: Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures. In: ICML (1). JMLR Workshop and Conference Proceedings, vol. 28, pp. 115–123. JMLR.org (2013)
10. Chunhong, Z., Licheng, J.: Automatic parameters selection for SVM based on GA. In: Fifth World Congress on Intelligent Control and Automation, WCICA 2004, vol. 2, pp. 1869–1872. IEEE (2004)
11. Friedrichs, F., Igel, C.: Evolutionary tuning of multiple SVM parameters. Neurocomputing **64**, 107–117 (2005)
12. Guyon, I., et al.: A brief review of the ChaLearn AutoML challenge: any-time any-dataset learning without human intervention. In: AutoML@ICML. JMLR Workshop and Conference Proceedings, vol. 64, pp. 21–30. JMLR.org (2016)
13. Guyon, I., et al.: Design of the 2015 ChaLearn AutoML challenge. In: IJCNN, pp. 1–8. IEEE (2015)

14. Frank, E., Hall, M.A., Holmes, G., Kirkby, R., Pfahringer, B.: WEKA - a machine learning workbench for data mining. In: Maimon, O., Rokach, L. (eds.) The Data Mining and Knowledge Discovery Handbook, pp. 1305–1314. Springer, Cham (2005). https://doi.org/10.1007/0-387-25465-X_62
15. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In: KDD, pp. 847–855. ACM (2013)
16. Kotthoff, L., Thornton, C., Hoos, H.H., Hutter, F., Leyton-Brown, K.: Auto-WEKA 2.0: automatic model selection and hyperparameter optimization in WEKA. J. Mach. Learn. Res. **18**, 25:1–25:5 (2017). http://jmlr.org/papers/v18/16-261.html
17. Olson, R.S., Bartley, N., Urbanowicz, R.J., Moore, J.H.: Evaluation of a tree-based pipeline optimization tool for automating data science. In: GECCO, pp. 485–492. ACM (2016)
18. Komer, B., Bergstra, J., Eliasmith, C.: Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In: ICML Workshop on AutoML (2014)
19. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J.T., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: NIPS, pp. 2962–2970 (2015)
20. de Sá, A.G.C., Pinto, W.J.G.S., Oliveira, L.O.V.B., Pappa, G.L.: RECIPE: a grammar-based framework for automatically evolving classification pipelines. In: McDermott, J., Castelli, M., Sekanina, L., Haasdijk, E., García-Sánchez, P. (eds.) EuroGP 2017. LNCS, vol. 10196, pp. 246–261. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55696-3_16
21. Lourenço, N., Pereira, F.B., Costa, E.: Unveiling the properties of structured grammatical evolution. Genet. Program. Evolvable Mach. **17**(3), 251–289 (2016). https://doi.org/10.1007/s10710-015-9262-4
22. O'Neill, M., Ryan, C.: Grammatical evolution. IEEE Trans. Evol. Comput. **5**(4), 349–358 (2001)
23. Keijzer, M., O'Neill, M., Ryan, C., Cattolico, M.: Grammatical evolution rules: the mod and the bucket rule. In: Foster, J.A., Lutton, E., Miller, J., Ryan, C., Tettamanzi, A. (eds.) EuroGP 2002. LNCS, vol. 2278, pp. 123–130. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45984-7_12
24. Thorhauer, A., Rothlauf, F.: On the locality of standard search operators in grammatical evolution. In: Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J. (eds.) PPSN 2014. LNCS, vol. 8672, pp. 465–475. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10762-2_46
25. Dua, D., Graff, C.: UCI machine learning repository (2017). http://archive.ics.uci.edu/ml
26. Chen, X., et al.: Gene expression patterns in human liver cancers. Mol. Biol. Cell **13**(6), 1929–1939 (2002)
27. Chowdary, D., et al.: Prognostic gene expression signatures can be measured in tissues collected in RNAlater preservative. J. Mol. Diagn. **8**(1), 31–39 (2006)
28. Wan, C., Freitas, A.A., De Magalhães, J.P.: Predicting the pro-longevity or anti-longevity effect of model organism genes with new hierarchical feature selection methods. IEEE/ACM Trans. Comput. Biol. Bioinform. (TCBB) **12**(2), 262–275 (2015)