Automated Selection and Configuration of Multi-Label Classification Algorithms with Grammar-based Genetic Programming

Alex G. C. de Sá[1], Alex A. Freitas[2], and Gisele L. Pappa[1]

[1] Computer Science Department, Universidade Federal de Minas Gerais, Brazil
[2] School of Computing, University of Kent, UK
alexgcsa@dcc.ufmg.br, A.A.Freitas@kent.ac.uk, glpappa@dcc.ufmg.br

Abstract. This paper proposes Auto-MEKA$_{GGP}$, an Automated Machine Learning (Auto-ML) method for Multi-Label Classification (MLC) based on the MEKA tool, which offers a number of MLC algorithms. In MLC, each example can be associated with one or more class labels, making MLC problems harder than conventional (single-label) classification problems. Hence, it is essential to select an MLC algorithm and its configuration tailored (optimized) for the input dataset. Auto-MEKA$_{GGP}$ addresses this problem with two key ideas. First, a large number of choices of MLC algorithms and configurations from MEKA are represented into a grammar. Second, our proposed Grammar-based Genetic Programming (GGP) method uses that grammar to search for the best MLC algorithm and configuration for the input dataset. Auto-MEKA$_{GGP}$ was tested in 10 datasets and compared to two well-known MLC methods, namely Binary Relevance and Classifier Chain, and also compared to GA-Auto-MLC, a genetic algorithm we recently proposed for the same task. Two versions of Auto-MEKA$_{GGP}$ were tested: a full version with the proposed grammar, and a simplified version where the grammar includes only the algorithmic components used by GA-Auto-MLC. Overall, the full version of Auto-MEKA$_{GGP}$ achieved the best predictive accuracy among all five evaluated methods, being the winner in six out of the 10 datasets.

Keywords: Automated Machine Learning (Auto-ML), Multi-Label Classification, Grammar-based Genetic Programming.

1 Introduction

The outgrowing popularity of machine learning algorithms and its indiscriminate use by practitioners who do not necessarily know the peculiarities of these methods have made the area of automated machine learning (Auto-ML) [3, 5, 6, 8, 14] more relevant than ever. The area of Auto-ML emerged to deal with the problem of how to select learning algorithms and their hyper-parameters to successfully solve a given ML problem. This problem is a hard one even for experts, which usually follow ad-hoc approaches to choose learning algorithms. In the majority of cases, such decisions are based on trial and error when testing different methods from the literature or on the recommendation of other experienced

data scientists. Additionally, the algorithm's hyper-parameters are rarely deeply explored to achieve the best algorithm's performance for the given problem.

This scenario makes many ML solutions biased, incomplete and inefficient. Auto-ML proposes to deal with these problems by customizing solutions (in terms of algorithms and configurations) to ML problems. Most Auto-ML systems proposed to date focus on generating sequences of steps to solve single label classification (SLC) problems [3, 5, 6, 8, 14]. The objective of classification is to learn models from data capable of expressing the relationships between a set of predictive attributes and a predefined set of class labels. In the case of SLC, each instance is associated to a single class label.

However, there is an increasing number of applications that require associating an example to more than one class label, including image and video annotation, gene function prediction, medical diagnosis and tag suggestion for text mining. For example, in the context of medical diagnosis, a patient can be associated to one or more diseases (e.g., diabetes, pancreatic cancer and high blood pressure) at the same time. This classification scenario is better known as multi-label classification (MLC) [15]. MLC is considered a more challenging problem than SLC. First, the algorithm needs to consider the label correlations (i.e., detecting if they exist or not) in order to learn a model that produces accurate classification results. Second, the limited number of examples for each class label in the dataset makes generalization harder, as the algorithm needs more examples to create a good model from such complex data.

In the same way that MLC is harder than SLC, we consider the Auto-ML task for MLC data more challenging than the Auto-ML task for SLC data. This is because of the higher difficulty to learn from multi-label data, the strain to evaluate the produced MLC models [15], and the computational cost involved. Despite these problems, we have recently proposed the first Auto-ML method to tackle MLC [2], here referred to as GA-Auto-MLC. The method is a simple real-coded genetic algorithm (GA) that performs a search in a very large (hierarchical) search space of many different types of MLC algorithms from the MEKA framework [12]. Although GA-Auto-MLC was effective in the experiments reported in [2] (with only three datasets), its solution encoding approach has two major drawbacks: it is cumbersome and it allows individuals representing impractical MLC algorithm configurations (in the sense that the MLC algorithms could have invalid configurations or take too long to run).

GA-Auto-MLC encodes solutions using a real-valued array to code a complex hierarchical structure representing the MLC algorithms and their hyper-parameters. Although the genotype is represented by a vector of a fixed predefined size, each position of the array can map to distinct components (essential functional parts) of MLC algorithms. In other words, the genes do not have any semantic meaning regarding the mapping to the phenotype. Because of that, when performing genetic operations (such as crossover and mutation), some operations are highly conservative (e.g., no changes occur in the phenotype after a mutation operation) while others highly destructive (e.g., abrupt changes occur in the phenotype after a mutation operation).

Aiming to address the aforementioned problems, this paper proposes a new evolutionary Auto-ML for MLC (based on the MEKA tool), namely Automated

MEKA (Auto-MEKA$_{GGP}$). Auto-MEKA$_{GGP}$ is a grammar-based genetic programming method [7] capable of handling the complex hierarchical nature of the MLC search space while avoiding the generation of invalid solutions. The method was conceived to explore a larger set of MLC algorithms and components when compared to GA-Auto-MLC. Auto-MEKA$_{GGP}$ optimizes the choice of an MLC algorithm and hyper-parameter settings to the target problem.

In order to evaluate its effectiveness, Auto-MEKA$_{GGP}$ was tested in 10 datasets and compared to two well-known MLC algorithms: Binary Relevance (BR) [15] and Classifier Chain (CC) [11]. Auto-MEKA$_{GGP}$ was also compared to GA-Auto-MLC, and all comparisons were based on a combination of several multi-label predictive accuracy measures [2, 15]. We run two versions of Auto-MEKA$_{GGP}$: a full version with our proposed grammar, and a simplified grammar version including only the components of GA-Auto-MLC. The results showed that Auto-MEKA$_{GGP}$ was the best method in terms of average rank, followed by its simplified version, and then GA-Auto-ML, BR and CC.

The remainder of this paper is organized as follows. Section 2 reviews related work on Auto-ML and MLC. Section 3 details the proposed method, while Section 4 presents and discusses the results obtained. Finally, Section 5 draws some conclusions and discusses directions of future work.

2 Related Work

Currently, Auto-ML methods [3, 5, 6, 8, 14] have been dealing with the optimization of complete ML pipelines. This means that, instead of just focusing on ML algorithms and their hyper-parameters, these methods are also concerned with other aspects of ML, such as data preprocessing (e.g., feature normalization or feature selection) and post-processing (e.g., classification probability calibration). Most methods proposed so far in the literature use as their search method either Bayesian optimization or evolutionary approaches.

Auto-WEKA [14] automates the process of selecting the best ML pipeline in WEKA [16], whereas Auto-SKLearn [5] optimizes the pipelines in Scikit-Learn [10]. Both methods implemented a random forest based version of a Bayesian optimization approach (i.e., Sequential Model-based Algorithm Configuration).

Evolutionary methods are also commonly used to perform this task. The Tree-Based Pipeline Optimization Tool (TPOT) [8], for instance, applies a canonical genetic programming (GP) algorithm to search for the most appropriate ML pipeline in the Scikit-Learn library. Considering a different evolutionary approach, the Genetic Programming for Machine Learning method (GP-ML) [6] uses a strongly typed genetic programming (STGP) method to restrict the Scikit-Learn pipelines in such a way that they are always meaningful from the machine learning point of view. Finally, the REsilient ClassifIcation Pipeline Evolution method (RECIPE) [3] adopts a grammar-based genetic programming (GGP) method to search for Scikit-Learn pipelines. It uses a grammar to organizes the knowledge acquired from the literature on how successful ML pipelines look like. The grammar avoids the generation of invalid pipelines, and can also speed up the search.

All Auto-ML methods previously discussed were designed to solve the conventional *single-label* classification task. By contrast, we propose Auto-MEKA$_{GGP}$, a grammar-based genetic programming method to solve the Auto-ML task for *multi-label* data. Auto-MEKA$_{GGP}$ overcomes the major drawbacks of our previously proposed GA-Auto-MLC method [2], being able to properly handle the complex hierarchical nature of the MLC search space. It is important to point out that in this paper we focus only on algorithms and hyper-parameters (not pipelines), as the MLC search space is much bigger than the SLC search space.

Most works in the MLC literature fall into one of two approaches [15]: problem transformation (PT) and algorithm adaptation (AA). While PT creates algorithms that transform the multi-label dataset (task) into one or more single-label classification tasks (making it possible to use any SLC algorithm), AA adapts traditional single-label classification algorithms to handle multi-label data.

Among the many MLC algorithms in the literature, it is worth mentioning: Binary Relevance (BR), which learns $Q = |L|$ independent binary classifiers, one for each label in the label set L; Label Powerset (LP), which creates a single class for each unique set of labels that exists in a multi-label training set; and Classifier Chain (CC), which extends the BR method by chaining the Q binary classifiers (also one for each label), where the attribute space of each link in the chain is increased with the classification outputs of all previous links. For more details about MLC algorithms, see [1, 15].

Given the very large variety of MLC algorithms in the literature — each one having its own assumptions or biases — it is clear that selecting the best MLC algorithm for a dataset is a hard task, and the use of Auto-ML is fully justified. This is because different algorithms' assumptions can lead to different predictive performances, depending on the characteristics of the dataset and the algorithms. For instance, when the BR method is selected, the label correlations are disregarded, which is beneficial for some types of datasets. However, considering the label correlations is essential for some other datasets, which makes LP and CC methods better choices. Hence, it is important to identify these patterns and map specific algorithms (with hyper-parameters) to specific datasets.

3 Automatically Selecting Algorithms and Hyper-Parameters for Multi-Label Classification

This section presents Automated MEKA (Auto-MEKA$_{GGP}$), a method conceived to automatically select and configure MLC algorithms in the MEKA tool [12]. Auto-MEKA$_{GGP}$ relies on a grammar-based genetic programming (GGP) search to select the best MLC algorithm and its associated hyper-parameters to a given dataset. The GGP search naturally explores the hierarchical nature of the problem, a missing feature of our previous method [2].

As shown in Figure 1, Auto-MEKA$_{GGP}$ receives as input an MLC dataset (with the attribute space X_F with F features and the Q class labels, L_1 to L_Q) and a grammar describing the (hierarchical) search space of MLC algorithms and their hyper-parameters. The grammar directly influences the search, as each individual created by the GGP is based on its production rules, which guarantees

that all individuals are valid. In other words, the MLC grammar defines the search space and how the individuals are created and modified (see Section 3.1).

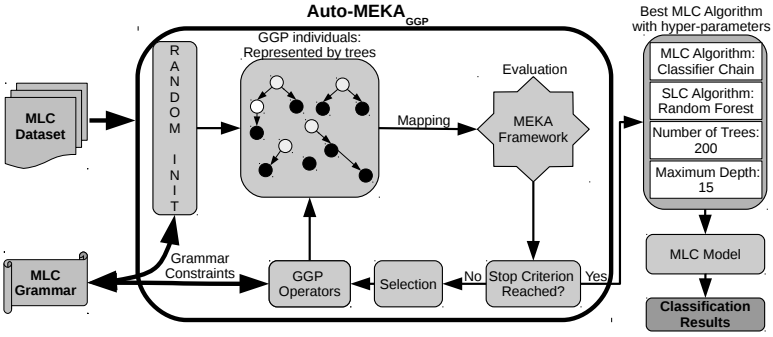


Fig. 1. The proposed method to select and configure MLC algorithms.

Auto-MEKA$_{\text{GGP}}$ works as follows. First, it creates an initial population of individuals (trees representing MLC algorithms) by choosing at random valid rules from the grammar (see Section 3.1), generating a derivation tree. Next, an iterative process starts. First, a mapping of each derivation tree to a specific MLC algorithm is performed. The individuals are evaluated by running the algorithm they represent within the MEKA tool on the input (see Section 3.2). Different MLC measures are taken into account to assess the individuals' quality, i.e., the fitness function. Next, Auto-MEKA$_{\text{GGP}}$ checks if a search stopping criterion is satisfied (e.g., a fixed number of iterations or a quality criteria). If this criterion is not satisfied, Auto-MEKA$_{\text{GGP}}$ selects individuals by using tournament selection. Next, the GGP operators (i.e., Whigham's crossover and mutation [7]) are applied on the selected individuals to create a new population. These operators also respect the grammar constraints, ensuring that the produced individuals represent valid solutions. This process goes on until the stopping criterion is satisfied. At the end of the evolution, the best individual (an MLC algorithm with its hyper-parameters) is returned, and its model is built from the full training set and evaluated in the test set (which was not accessed during the evolution), in order to measure the predictive performance of the returned individual.

It is worth noting that Auto-MEKA$_{\text{GGP}}$ was implemented using EpochX [9], an open source genetic programming framework, and is available for download[1].

3.1 Grammar: A Formal Description of the MLC Search Space

This section describes the grammar used to specify the search space of our proposed Auto-MEKA$_{\text{GGP}}$ method. The Auto-MEKA$_{\text{GGP}}$'s grammar was created based on MEKA, which is a multi-label extension to WEKA [16], and hence in-

[1] Code and documentation are available at: https://github.com/laic-ufmg/automlc/

```
<Start> ::= (<MLC-PT> | <MLC-AA> | <META-MLC-LEVEL> ) <Pred-Tshd>
<MLC-PT> ::= <ALGS-PT> <ALGS-SLC>
<ALGS-SLC> ::= <ALG-TYPE> | <META-SLC1> <ALG-WEIGHTED-TYPE> | <META-SLC2> <ALG-RANDOM-TYPE> |
               (<META-SLC3> | <META-SLC4> [<ALG-TYPE>] [<ALG-TYPE>] [<ALG-TYPE>] <ALG-TYPE>) <ALG-TYPE>
                                        #META-SLC 1-4='Meta SLC algorithms with different constraints'
                                        #ALG-WEIGHTED-TYPE='Defining SLC weighted algorithms'
                                        #ALG-RANDOM-TYPE='Defining randomizable SLC algorithms'
<ALG-TYPE> ::= [<ASC>] (<Trees> | <Rules> | <Lazy> | <Functions> | <Bayes> | <Exceptions>)
...                                     #ALG-TYPE='Types of SLC algorithms'
                                        #ALGS-PT 1-4='PT methods with different constraints'
<ALGS-PT> ::= <ALGS-PT1> | <ALGS-PT2> | <ALGS-PT3> | <ALGS-PT4>
<ALGS-PT1> ::= BR | CC | LP          #BR='Binary Relevance', CC='Classifier Chain', #LP='Label Powerset'
<MLC-AA> ::= <ML-DBPNN> <ML-BPNN> | <ML-BPNN>    #ML-BPNN='Multi-Label Back Propagation Neural Network'
...                                     #ML-DBPNN= Deep ML-BPNN
<META-MLC-LEVEL> ::= <META-MLC1> | <META-MLC2> | <META-MLC3> | <META-MLC4> | MBR BR <ALGS-SLC>
                                        #MBR='BR method stacked with feature outputs'
...                                     #META-MLC 1-4='Meta MLC algorithms with different constraints'
<Pred-Tshd> ::= PCut1 | PCutL | RANDOM-REAL(0.001, 0.999)   #Pred-Tshd='prediction threshold'
```

Fig. 2. An excerpt of the proposed grammar for multi-label classification.

cludes most of its algorithms. MEKA has a large variety of algorithms, focusing mainly on problem transformation methods.

We first performed a deep study of the MLC search space in MEKA: the algorithms and their hyper-parameters, the constraints associated with different hyper-parameter settings, the hierarchical nature of operations performed by problem transformation algorithms and meta-algorithms, and other issues. The grammar includes 30 MLC algorithms, exploring most algorithms in MEKA. We let some algorithms aside because of their poor performance to solve the MLC task or because of errors when testing the algorithm for different types of data. The MLC algorithms were divided into three types: problem transformation (PT), algorithm adaptation (AA) and meta-algorithms (Meta).

PT algorithms usually call the SLC algorithms to solve an MLC problem, transforming the given problem into one or various SLC problems. For these algorithms, we choose 30 SLC algorithms based on a robust method to select and configure algorithms in WEKA, i.e., Auto-WEKA [14]. On the other hand, AA methods do not need to transform the data in a preprocessing step, applying their learning process in a direct way. Finally, meta algorithms have the aforementioned MLC algorithms (PT or AA) as base algorithms, using the base classifiers' outputs in different ways to try to improve MLC performance. Considering these learning algorithms, their hyper-parameters, their dependencies and constraints, the search space of MLC algorithms has $(8.420 \times 10^{128}) + [(5.642 \times 10^{124}) \times Q] + [(1.755 \times 10^{113}) \times Q^2]$ possible MLC algorithm configurations, where Q is the number of labels of the input dataset. For more details about these possible algorithm configurations, see [1].

After studying this search space, we defined a grammar that encompasses the knowledge about MLC in MEKA, i.e., all algorithms, hyper-parameters and constraints. Formally, a grammar G is represented by a four-tuple $<N, T, P, S>$, where N represents a set of non-terminals, T a set of terminals, P a set of production rules and S (a member of N) the start symbol.

Figure 2 presents a sample of our proposed grammar. The complete version of the MLC grammar is specified in [1] and the implemented grammar (i.e., for EpochX) is also available online[2]. The proposed grammar has 138 production rules, in a total of 137 non-terminals and 230 terminals. It uses the Backus Naur Form (BNF), where each production rule has, for instance, the form <*Start*>::= <*Meta-Algorithm*><*AlgorithmA*> | <*AlgorithmB*> *ParamA*. Symbols wrapped in "< >" represent non-terminals, whereas terminals (such as ParamA) are not bounded by "< >". The special symbols "|", "[]" and "()" represent, respectively, a choice, an optional element and a set of grouped elements that should be used together. Additionally, the symbol "#" represents a comment in the grammar, i.e., it is ignored by the grammar's parser. The choice of one among all elements connected by "|" is made using a uniform probability distribution (i.e., all elements are equally likely to occur in an individual).

3.2 From Individual Representation to Individual Evaluation

Each individual is represented by a tree, derivated from the expansion of the production rules of the MLC grammar. The mapping process takes the terminals from the tree and constructs a valid MLC algorithm from them. Given the mapped MLC algorithm in MEKA (and WEKA), the fitness function measures how effective each algorithm is for the input dataset To do this, the training set is split into two parts: a learning set (80%) and a validation set (20%). We use a stratified sampling method [13] to split the training set. Each MLC algorithm creates an MLC model from the learning set and evaluates its predictive accuracy on the validation set, using the fitness function.

MLC algorithms are usually evaluated considering several measures [15]. Hence, we set the fitness function as the average of four of these MLC measures [2, 15]: Exact Match (EM), Hamming Loss (HL), F_1 Macro averaged by label (FM) and Ranking Loss (RL), as indicated in Equation 1:

$$Fitness = \frac{EM + (1 - HL) + FM + (1 - RL)}{4} \tag{1}$$

EM is a very strict evaluation metric, as it only takes the value one when the predicted label set is an exact match to the true label set for an example, and takes the value zero otherwise. HL counts how many times a label not belonging to the example is predicted, or a label belonging to the example is not predicted. FM is the harmonic mean between precision and recall, and its average is first calculated per label and, after that, across all the labels in the dataset. This metric is interesting because it accounts for different levels of class imbalance of the data. Finally, RL measures the number of times that irrelevant labels are ranked higher than relevant labels, i.e., it penalizes the label pairs that are reversely ordered in the ranking for a given example. All four metrics are within the $[0, 1]$ interval. However, the EM and FM measures should be maximized, whereas HL and RL should be minimized. Hence, HL and RL are subtracted from one in Equation 1 to make the search maximize the fitness function.

[2] The implementation of the grammar(s) for EpochX is available at: https://github.com/laic-ufmg/automlc/tree/master/PPSN

4 Experimental Results

This section presents the experimental results of the proposed method in 10 datasets from the KDIS (Knowledge and Discovery Systems) repository[3]. The datasets are presented in the first two columns of Table 1, where name of the dataset is followed by a three-tuple (M, F, Q), where M is the number of examples, F is the number of features, and Q is the number of labels.

Tests are performed with two different grammar versions: a simplified version[4] that matches the search space of GA-Auto-MLC [2] and a full version[5] corresponding to the complete set of MLC components defined in this paper. The simplified version (i.e., Auto-MEKA$_{\text{GGP(S)}}$) allows us to directly compare our results to those obtained by GA-Auto-MLC.

The two versions of Auto-MEKA$_{\text{GGP}}$ and GA-Auto-MLC were executed with the following parameters: 100 individuals evolved for at most 100 generations, tournament selection of size two, elitism of five individuals, and crossover and mutation probabilities of 0.9 and 0.1, respectively. If the best individual remains the same for over five generations and the algorithm has run for at least 20 generations, we stop the evolutionary process and return that best individual. The learning and validation sets are resampled from the training set every five generations in order to avoid overfitting. Additionally, we use time and memory budgets for each MLC algorithm (generated by the evolutionary Auto-MLC methods) of 450 seconds (7.5 minutes) and 2GB of RAM, respectively.

The algorithms produced are also compared to Binary Relevance (BR) and Classifier Chain (CC) methods. These two methods do not have hyper-parameters at the MLC level, but can use different SLC algorithms. We test them with 11 candidate algorithms [16]: Naïve Bayes (NB), Tree Augmented Naïve Bayes (TAN), Bayesian Network Classifier algorithm with a K2 search method (BNC-K2), Logistic Model Trees (LMT), Random Forest (RF), C4.5 (J48), Sequential Minimal Optimization (SMO), Multi-Layer Perceptron (MLP), K-Nearest Neighbors (KNN), PART and Logistic Regression (LR). All SLC algorithms use the default hyper-parameters, except for SMO which uses a Gaussian Kernel (with default hyper-parameters), and for KNN which searches for the best K value in the interval [1,20] by performing a leave-one-out procedure based on the learning and validation sets. Note that identical time and memory budgets were applied in this local search to provide a fair comparison.

We perform the experiments using a stratified five-fold cross-validation [13] with six repetitions varying Auto-MEKA$_{\text{GGP}}$'s random seed, resulting in 30 runs per dataset for each method. Table 1 presents the (average) results of fitness function (see Equation 1) in the test set followed by their standard deviations. For each dataset, the best average result is displayed in bold.

We use the well-known statistical approach proposed by Demšar [4] to compare different methods, using an adapted Friedman test followed by a Nemenyi post hoc test with significance level of 0.05. The last two rows of Table 1 show the average value and the average rank for each method.

[3] The datasets are available at: http://www.uco.es/kdis/mllresources/

[4] Available at: https://github.com/laic-ufmg/automlc/tree/master/PPSN/AutoMEKAS.bnf

[5] Available at: https://github.com/laic-ufmg/automlc/tree/master/PPSN/AutoMEKA.bnf

Table 1. The characteristics of the datasets, and the comparison for the versions of Auto-MEKA$_{GGP}$ and the baseline methods in the test set as to the fitness function.

Dataset	(M,F,Q)	Auto-MEKA *GGP*	Auto-MEKA *GGP(S)*	GA-Auto-MLC	BR	CC
Flags	(194,18,7)	**0.606 (0.02)**	0.598 (0.02)	0.603 (0.03)	0.582 (0.02)	0.590 (0.04)
Scene	(2407,294,6)	**0.837 (0.01)**	0.830 (0.01)	0.826 (0.01)	0.824 (0.01)	0.787 (0.02)
Birds	(645,260,19)	**0.724 (0.02)**	0.718 (0.02)	0.722 (0.01)	0.715 (0.03)	0.657 (0.02)
Yeast	(2417,103,14)	0.567 (0.01)	**0.568 (0.01)**	0.565 (0.01)	0.566 (0.00)	0.552 (0.01)
GPosPse	(519,440,4)	**0.734 (0.04)**	0.729 (0.04)	0.721 (0.03)	0.700 (0.04)	0.697 (0.04)
CHD_49	(555,49,6)	**0.554 (0.02)**	0.549 (0.02)	0.550 (0.02)	0.540 (0.02)	0.524 (0.02)
WTQlty	(1060,16,14)	0.521 (0.01)	0.522 (0.01)	**0.524 (0.01)**	0.523 (0.02)	0.483 (0.01)
Emotions	(593,72,6)	0.668 (0.02)	**0.676 (0.02)**	0.674 (0.01)	0.666 (0.02)	0.627 (0.02)
Reuters	(294,1000,6)	0.473 (0.04)	0.475 (0.04)	**0.476 (0.05)**	0.469 (0.04)	0.457 (0.04)
Genbase	(662,1186,27)	**0.941 (0.01)**	0.938 (0.01)	0.938 (0.01)	0.887 (0.10)	0.934 (0.01)
Average Values		0.663	0.660	0.660	0.647	0.631
Average Ranks		1.800	2.250	2.250	3.900	4.800

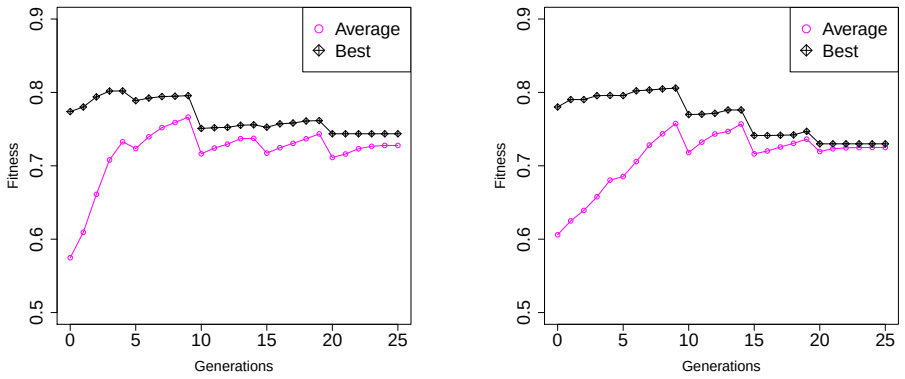
As shown in Table 1, Auto-MEKA$_{GGP}$ has achieved the best (lowest) average rank (based on the fitness measure), followed by Auto-MEKA$_{GGP(S)}$ and GA-Auto-MLC, which had the same rank. BR and CC presented the worst performances. There was no statistically significant difference between the performances of the two versions of Auto-MEKA$_{GGP}$ and GA-Auto-MLC. However, Auto-MEKA$_{GGP}$ was the best method in six out of the 10 datasets in Table 1, whilst Auto-MEKA$_{GGP(S)}$ and GA-Auto-MLC were each the best in only two datasets. Finally, both versions of Auto-MEKA$_{GGP}$ (and, also, GA-Auto-MLC) performed statistically better than BR and CC, well-known MLC methods.

4.1 Evolutionary Behavior

This section compares the evolutionary behaviors of Auto-MEKA$_{GGP}$ and GA-Auto-MLC. We did not include Auto-MEKA$_{GGP(S)}$ in this analysis because its results were not significantly better than those achieved by GA-Auto-MLC.

Figures 3(a) and 3(b) illustrate the fitness evolution of the best individuals of the population and the average fitness of the population of individuals of Auto-MEKA$_{GGP}$ and GA-Auto-MLC for the dataset GPosPse. All curves consider the mean of the only 10 runs (out of 30) with the same final number of generations (25). This dataset was chosen because it shows a situation where Auto-MEKA$_{GGP}$ is clearly better than GA-Auto-MLC, but the evolution curves are similar for other datasets. Note that the fitness values of the individuals can decrease or increase from one generation to another due to training data resampling.

Observe that Auto-MEKA$_{GGP}$'s population converges faster than GA-Auto-MLC's one. This may be due to the lack of semantic meaning of the genes in GA-Auto-ML's individuals, so a GA-Auto-MLC's individual can change severely from one generation to another by performing crossover and mutation. This is less likely to happen in Auto-MEKA$_{GGP}$ as the grammar restricts the GGP operations, which explains why the produced individuals converge quickly.



(a) Auto-MEKA$_{\text{GGP}}$'s behavior. (b) GA-Auto-MLC's behavior.

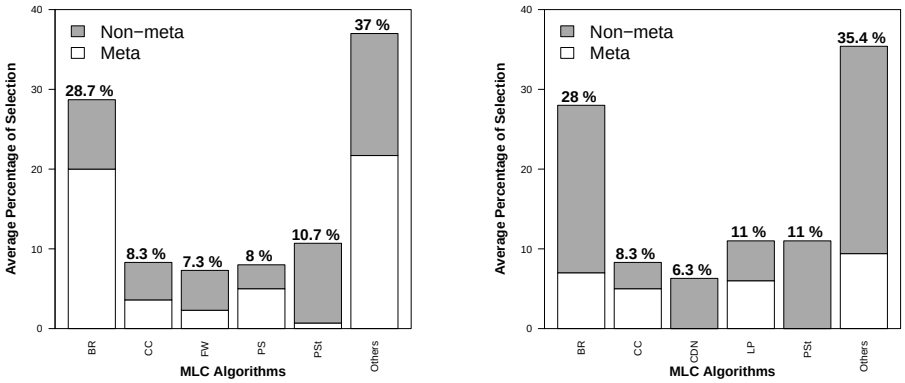
Fig. 3. Evolution of fitness values for the dataset GPosPse.

4.2 The Diversity of the Selected MLC Algorithms

This section analyzes the diversity of the MLC algorithms selected by two evolutionary Auto-ML methods: Auto-MEKA$_{\text{GGP}}$ and GA-Auto-ML. We focus only on the selected MLC algorithms (the "macro-components" of the Auto-ML methods), and not on their selected parameter settings (the "micro-components"), to simplify the analysis. We do not report results for Auto-MEKA$_{\text{GGP(S)}}$ because again the full version of this method, Auto-MEKA$_{\text{GGP}}$, obtained better results, as discussed earlier. Analyzing the MLC algorithms selected by Auto-MEKA$_{\text{GGP}}$ and GA-Auto-ML can help us to better understand the results of Table 1, giving an idea of how the choice of an MLC algorithm influences the performance of these two Auto-ML methods.

Figures 4(a) and 4(b) present the bar plots to analyze the percentage of selection of MLC algorithms for the Auto-ML methods. For the full details about each MLC algorithm, see [1]. In these figures, we have for each MLC algorithm a (gray/white) bar, representing the average percentage of selection over all the 300 runs: 10 datasets times 30 independent runs per dataset (5 cross-validation folds times 6 single runs with different random seeds). These percentages rely on two cases: (i) when the traditional MLC algorithm is solely selected; (ii) when the traditional MLC algorithm is selected together with a MLC meta-algorithm. To emphasize these two cases, the bar for each traditional MLC algorithm is divided into two parts, with sizes proportional to the percentage of selection as a standalone algorithm (in gray color) and the percentage of selection as part of a meta-algorithm (in white color).

BR was the traditional MLC algorithm most frequently selected (in about 30% of all runs) by both Auto-ML methods. Besides, Classifier Chain (CC), Four-Class Pairwise Classification (FW), and Pruned Sets with and without threshold (PS and PSt) were selected in total in 34.3% of all runs by Auto-MEKA$_{\text{GGP}}$;



(a) Auto-MEKA$_{\text{GGP}}$. (b) GA-Auto-MLC.

Fig. 4. Barplots for the MLC algorithms' selection over all the 300 runs.

whilst CC, Conditional Dependency Networks (CDN), LP and PSt were selected in total in 36.6% of all runs by GA-Auto-MLC. Note that the MLC algorithms most frequently selected by Auto-MEKA$_{\text{GGP}}$ and GA-Auto-ML are broadly similar, which suggests that Auto-MEKA$_{\text{GGP}}$'s superior performance is partly due to a better exploration of the space of hyper-parameter settings of those most successful MLC algorithms. Finally, we observed that Auto-MEKA$_{\text{GGP}}$ selected meta MLC algorithms in 53.3% of all runs, whilst GA-Auto-MLC selected meta MLC algorithms in only 27.4% of all runs.

5 Conclusions and Future Work

This paper introduced Auto-MEKA$_{\text{GGP}}$, a new grammar-based genetic programming method for automatically selecting the best Multi-Label Classification (MLC) algorithm and its hyper-parameter settings for an input dataset. Auto-MEKA$_{\text{GGP}}$ uses a grammar representing prior knowledge about MLC algorithms, restricting its search space to valid solutions.

Auto-MEKA$_{\text{GGP}}$ was compared to two well-known MLC algorithms – Binary Relevance (BR) and Classifier Chain (CC) – and to GA-Auto-ML, a GA we recently proposed to solve this task [2]. We tested Auto-MEKA$_{\text{GGP}}$ with the full version of the proposed grammar and with a simplified grammar version which has the same search space (candidate MLC algorithms and their hyper-parameters) as GA-Auto-ML. Overall, the full version of Auto-MEKA$_{\text{GGP}}$ obtained the highest predictive accuracy among all five tested methods, being the winner in six out of the 10 datasets. Also, both versions of Auto-MEKA$_{\text{GGP}}$, as well as GA-Auto-ML, obtained statistically significantly higher accuracies than BR and CC.

In future work we plan to extend Auto-MEKA$_{GGP}$ to search for MLC pipelines too. This means to search for the best combination of MLC algorithms, data preprocessing and post-processing methods, and their respective hyper-parameters.

Acknowledgments: This work has been partially funded by the EUBra-BIGSEA project by the European Commission under the Cooperation Programme (MCTI/ RNP 3rd Coordinated Call), Horizon 2020 grant agreement 690116. In addition, this work has been partially supported by the following Brazilian Research Support Agencies: CNPq, CAPES and FAPEMIG.

References

1. A. G. C. de Sá, A. A. Freitas, and G. L. Pappa. Multi-label classification search space in the MEKA software. Technical report, UFMG, 2018. Available at: https://github.com/laic-ufmg/automlc/tree/master/PPSN/MLC-SearchSpace.pdf.
2. A. G. C. de Sá, G. L. Pappa, and A. A. Freitas. Towards a method for automatically selecting and configuring multi-label classification algorithms. In *Proc. GECCO Companion*, pages 1125–1132, 2017.
3. A. G. C. de Sá, W. J. Pinto, L. O. Oliveira, and G. L. Pappa. RECIPE: A grammar-based framework for automatically evolving classification pipelines. In *Proc. of the EuroGP Conference*, pages 246–261, 2017.
4. J. Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, 2006.
5. M. Feurer, A. Klein, K. Eggensperger, et al. Efficient and robust automated machine learning. In *Proc. of the International Conference on Neural Information Processing Systems*, pages 2755–2763, 2015.
6. T. Křen, M. Pilát, and R. Neruda. Automatic creation of machine learning workflows with strongly typed genetic programming. *Int. J. Artif. Intell. Tools*, 26(5):1–24, 2017.
7. R. Mckay, N. Hoai, P. Whigham, Y. Shan, and M. O'Neill. Grammar-based genetic programming: a survey. *Genet. Program Evolv. Mach.*, 11(3):365–396, 2010.
8. R. Olson, N. Bartley, R. Urbanowicz, and J. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proc. GECCO*, pages 485–492, 2016.
9. F. Otero, T. Castle, and C. Johnson. EpochX: genetic programming in Java with statistics and event monitoring. In *Proc. GECCO Companion*, pages 93–100, 2012.
10. F. Pedregosa, G. Varoquaux, A. Gramfort, et al. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.*, 12:2825–2830, 2011.
11. J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. *Machine Learning*, 85(3):333–359, 2011.
12. J. Read, P. Reutemann, B. Pfahringer, and G. Holmes. MEKA: A multi-label/multi-target extension to WEKA. *J. Mach. Learn. Res.*, 17(21):1–5, 2016.
13. K. Sechidis, G. Tsoumakas, and I. Vlahavas. On the stratification of multi-label data. In *Proc. of the ECML-PKDD Conference*, pages 145–158, 2011.
14. C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proc. of the ACM SIGKDD Conference*, pages 847–855, 2013.
15. G. Tsoumakas, I. Katakis, and I. Vlahavas. Mining multi-label data. In *Data Mining and Knowledge Discovery Handbook*, pages 667–685, 2010.
16. I. Witten, E. Frank, M. A. Hall, and C. Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 4th edition, 2016.