Fitness Landscape Analysis of Automated Machine Learning Search Spaces

Cristiano G. Pimenta[1(✉)], Alex G. C. de Sá[1], Gabriela Ochoa[2], and Gisele L. Pappa[1]

[1] Computer Science Department, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil
{cgpimenta,alexgcsa,glpappa}@dcc.ufmg.br
[2] University of Stirling, Stirling, UK
gabriela.ochoa@stir.ac.uk

Abstract. The field of Automated Machine Learning (AutoML) has as its main goal to automate the process of creating complete Machine Learning (ML) pipelines to any dataset without requiring deep user expertise in ML. Several AutoML methods have been proposed so far, but there is not a single one that really stands out. Furthermore, there is a lack of studies on the characteristics of the fitness landscape of AutoML search spaces. Such analysis may help to understand the performance of different optimization methods for AutoML and how to improve them. This paper adapts classic fitness landscape analysis measures to the context of AutoML. This is a challenging task, as AutoML search spaces include discrete, continuous, categorical and conditional hyperparameters. We propose an ML pipeline representation, a neighborhood definition and a distance metric between pipelines, and use them in the evaluation of the fitness distance correlation (FDC) and the neutrality ratio for a given AutoML search space. Results of FDC are counter-intuitive and require a more in-depth analysis of a range of search spaces. Results of neutrality, in turn, show a strong positive correlation between the mean neutrality ratio and the fitness value.

Keywords: Fitness landscape analysis · Automated Machine Learning · Fitness distance correlation · Neutrality

1 Introduction

The recent hype on machine learning (ML) and its application to a wide range of problems that are close to the general public has increased the interest in the area and, consequently, the number of people using ML to solve a wide range of problems [24]. However, the performance of an ML solution to a specific learning problem depends heavily on the choice of data preprocessing and learning algorithms, as well as on their hyperparameters. Although the choice of an ML solution can be manually made, this is a hard and not effective process.

L. Paquete and C. Zarges (Eds.): EvoCOP 2020, LNCS 12102, pp. 114–130, 2020.
https://doi.org/10.1007/978-3-030-43680-3_8

Considering the number of ML algorithms, combinations among them and their associated hyperparameters, the number of choices can grow exponentially. Furthermore, manual tuning requires an inherent expertise in the choice of methods and the possible values of their hyperparameters [24]. Hence, it is a challenging task for people who have little knowledge in ML.

The area of *Automated Machine Learning* (AutoML) has emerged as a solution to the aforementioned issue, becoming very popular over the past decades [8]. Its main objective is to automate the process of recommending or creating complete machine learning pipelines to any dataset without requiring deep user knowledge on the learning task itself [8]. A machine learning pipeline is a sequence of tasks to follow when performing data analysis in a specific dataset. It can include preprocessing steps (e.g., data cleaning, data discretization and feature selection [23]), a machine learning model (such as a classifier or a regressor), and postprocessing steps that may help to combine the results of several ML models (for instance, a voting method [23]).

Several optimization methods have been proposed to solve the problem of automatically generating ML pipelines, including those based on Bayesian optimization (e.g., Auto-WEKA and AutoSKLearn), evolutionary search (e.g., RECIPE and TPOT), multi-fidelity optimization (e.g., Hyperband) and hierarchical planing (e.g., ML-Plan) [8,24]. However, there is not a single one that seems to outperform all the others and, in most cases, very similar results are obtained by different methods.

AutoML methods based on optimization techniques rely on two main components: a search space and an optimization method. The search space comprises the main building blocks (e.g., the preprocessing methods, the learning models, the postprocessing approaches and their associated hyperparameters) from previously designed ML pipelines. The optimization method is responsible for finding the best combinations of ML components to build the most effective pipelines according to a quality metric to a given dataset.

There is still very little knowledge on how the characteristics of the search space impact AutoML methods. These search spaces are difficult to analyze, as they include discrete, continuous, categorical and conditional variables [8]. A better understanding of AutoML search spaces can help to explain the performance of existing algorithms and lead to the development of new ones, designed to explore the peculiarities of these spaces [14].

One way to analyze the characteristics of the search spaces is through fitness landscape analysis (FLA) [19]. The fitness landscape of a problem is given by the values of fitness obtained by all possible solutions present in the search space. The idea of FLA methods is to gain a better understanding of algorithm performance on a related set of problem instances, creating an intuitive understanding of how a heuristic algorithm explores the fitness landscape. However, as AutoML search spaces contain mixed types of variables, performing FLA in this case is more challenging because the notion of neighborhood or distance function needed by FLA metrics is not straightforward.

Fitness landscape analysis of algorithm configuration and machine learning pipeline generation is still in its early stages [7,15]. In this paper, we propose a way of measuring the distance between machine learning pipelines and adapt typical FLA metrics to the complex search spaces of AutoML. We then adapt the fitness distance correlation (FDC) and the neutrality ratio metrics for AutoML search spaces. Results of FDC are initially counter-intuitive and require a more in-depth analysis of a range of search spaces. Results of neutrality, in turn, show a strong positive correlation between the mean neutrality ratio and the fitness value. A next step is to investigate whether this is beneficial or detrimental to different search methods.

2 Problem Definition

Before defining the fitness landscape of an AutoML problem, we formally define the problem itself. AutoML can be cast as the *Combined Algorithm Selection and Hyperparameter optimization* (CASH) problem [6,20]. Given a set $\mathcal{A} = \{A^{(1)}, A^{(2)}, \ldots, A^{(k)}\}$ of learning algorithms, where each algorithm $A^{(j)}$ has a hyperparameter space $\Lambda^{(j)}$, the CASH problem is defined in Eq. 1. In its original formulation [20], CASH is defined as a minimization problem. Here we cast it as a maximization problem, replacing the loss function with a gain function.

$$A^*_{\lambda^*} \in \underset{A^{(j)} \in \mathcal{A}, \lambda \in \Lambda^{(j)}}{\operatorname{argmax}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{G}(A^{(j)}_{\lambda}, \mathcal{D}^{(i)}_{\text{train}}, \mathcal{D}^{(i)}_{\text{valid}}), \tag{1}$$

where $\mathcal{G}(A^{(j)}_{\lambda}, \mathcal{D}^{(i)}_{\text{train}}, \mathcal{D}^{(i)}_{\text{valid}})$ is the gain achieved when a learning algorithm A, with hyperparameters λ, is trained and validated on disjoint training and validation sets $\mathcal{D}^{(i)}_{\text{train}}$ and $\mathcal{D}^{(i)}_{\text{valid}}$, respectively, on each partition $1 \leq i \leq k$ of a k-fold cross-validation procedure. The main idea of this paper is to analyze the characteristics of the search space of algorithms $\mathcal{A}$ and the hyperparameters $\lambda^{(j)}$ of each $A_j \in \mathcal{A}$.

3 AutoML Fitness Landscape

Stadler [19] defines a fitness landscape as having three components: (i) a set X of configurations; (ii) a notion $\mathcal{X}$ of neighborhood or distance on X, and (iii) a fitness function $f : X \to \mathbb{R}$. The set X of configurations and the neighborhood definition $\mathcal{X}$ define the configuration space of the problem. Depending on $\mathcal{X}$, one fitness function can be associated with several different fitness landscapes [14]. The next sections discuss these three components in the context of AutoML.

3.1 Configurations

The first component of a fitness landscape is a set X of configurations. In the case of the AutoML problem tackled in this paper, X corresponds to all valid

classification machine learning pipelines that can be generated to solve a given problem. An ML pipeline can be defined as the sequence of algorithms that transform a feature vector $\vec{\boldsymbol{x}} \in \mathbb{X}^d$ (with d dimensions) into a target vector $\vec{\boldsymbol{y}} \in \mathbb{Y}$, which contains discrete values (i.e., class labels) for classification problems [24].

Machine Learning Pipelines: A typical machine learning pipeline is composed of preprocessing steps (e.g., data cleaning and feature selection), a machine learning modeling step (e.g., a classification or regression algorithm), and some postprocessing steps that may help to combine the results of several models [23].

For the sake of simplicity, in this first analysis we consider only classification pipelines composed of up to three preprocessing algorithms and one classifier, without any postprocessing steps. The pipelines are generated by creating derivation trees from a proposed context-free grammar (CFG). One of the benefits of using CFGs [18] to represent the AutoML search spaces is that they organize prior knowledge (from specialists) about the problem, properly guiding the optimization process. In addition, the grammar also gives flexibility in the definition of the search space, as the grammar rules can be modified anytime. Finally, the grammar can introduce semantics along with its syntax, possibly allowing the evaluation of the complexity of the search space.

The grammar defines the order of the preprocessing algorithms and guarantees a classification algorithm is always present in a pipeline. The search space the grammar defines is composed of 18 preprocessing and 23 classification algorithms. Given these algorithms an their associated hyperparameters, the grammar contains 148 terminal symbols and 128 non-terminal symbols and production rules, generating a search space with an estimated size of $7.88\text{e}9(feat-1)^2 + 1.62\text{e}18(feat-1) + 1.05\text{e}13$ pipeline configurations, where $feat$ is the number of features of the dataset[1]. The complete grammar and other supplementary material are available online[2].

An example of a pipeline, which is a derivation tree from the grammar, is shown in Fig. 1, where algorithm names correspond to tree nodes with sharp edges. Hyperparameter names are represented as rounded rectangles with dashed lines, whereas their values correspond to the ellipses. In this paper, pipelines are initialized at random by uniformly choosing production rules from the grammar.

3.2 Neighborhood and Distance Between Pipelines

The second component of a fitness landscape is a notion $\mathcal{X}$ of neighborhood or a distance metric between the elements of the set X of configurations, described in Sect. 3.1. Considering the complexity of the AutoML search space, which contains categorical, discrete, continuous and conditional parameters, and the lack of literature on the analysis of such spaces, we propose a simple neighborhood

[1] When determining the search space size, for continuous hyperparameters, we simplify and always consider 100 values, regardless of the size of the interval.

[2] https://cgpimenta.github.io/EvoCOP2020_CGPimenta/.

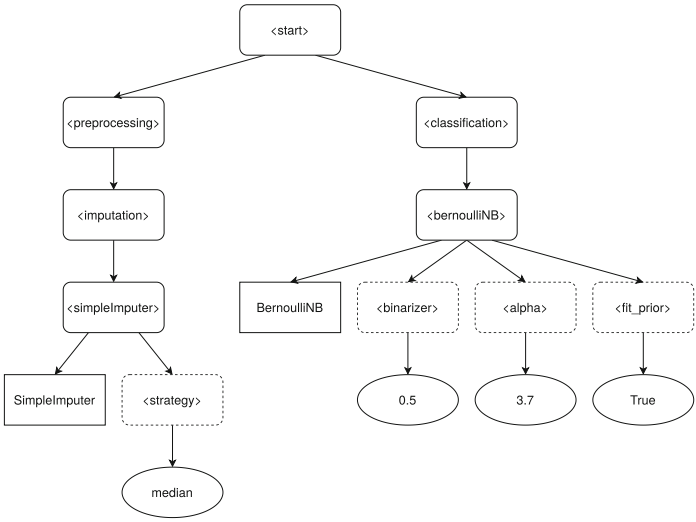


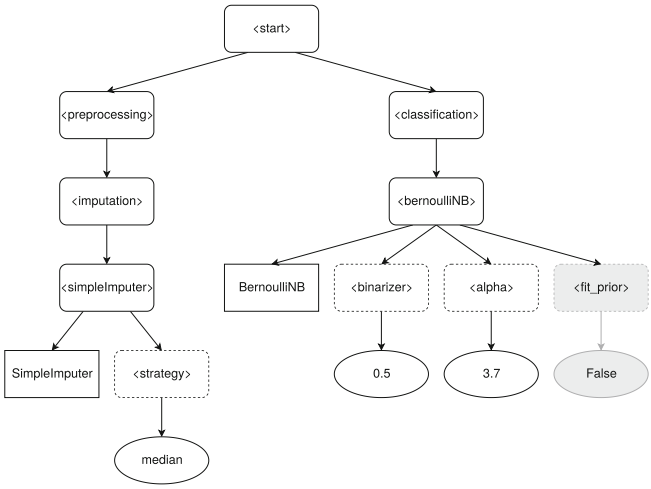
Fig. 1. Tree representation of a machine learning pipeline.

definition and a distance metric between our tree-based pipelines, adapted from the metric proposed by Ekárt & Németh for genetic programs [5].

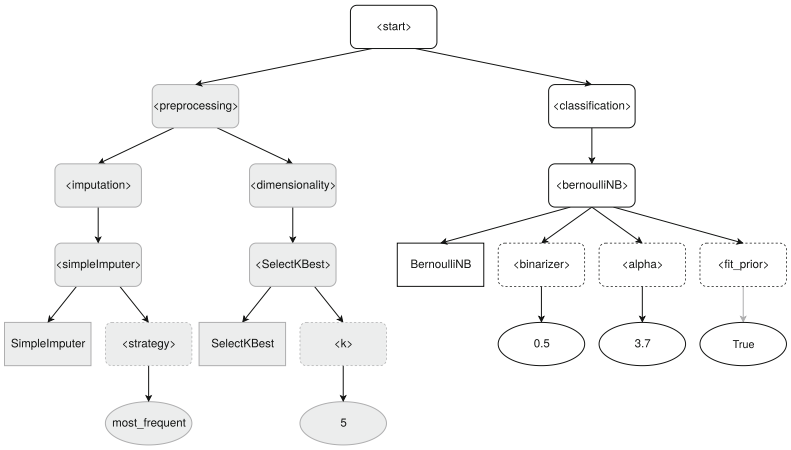
Neighborhood Definition: We defined the neighborhood $N(s)$ of machine learning pipeline s as the set that contains all trees that result from the application of a mutation operator on a random node of the tree (i.e., the selected node is replaced by another component generated by its parent node on the tree). Figure 2 shows two neighbors of the pipeline from Fig. 1. Grey subtrees indicate where the mutation operation was employed. Considering that changing an algorithm by another can have a much bigger impact than changing the value of a hyperparameter, we define the probability $p(x)$ of choosing a node x from tree T as the mutation point as a function that increases with the distance from the root. The exceptions are the terminal symbols (leaves of the tree), which are only changed when their parent node is selected. In order to do that, we give a weight $w(x)$ to each node that is directly proportional to the probability of choosing it, as defined in Eq. 2.

In this way, the root tree (*<start>* symbol) has weight 1, and the weight increases according to the level of the node in the tree. In our case, the preprocessing symbol has weight 2, the classification and preprocessing subgroups weight 3. The non-terminals representing algorithm names have weight 4 and those representing hyperparameter names, weight 5. $p(x)$ is then given by Eq. 3.

$$w(x) = \begin{cases} \text{tree level} & \text{if non-terminal symbol} \\ 0 & \text{if terminal symbol} \end{cases} \tag{2}$$



(a) Value of the `fit_prior` hyperparameter changed from `True` to `False`.



(b) Whole preprocessing subtree mutated.

Fig. 2. Two neighbors of the pipeline from Fig. 1. Grey subtrees indicate mutation points.

$$p(x) = \frac{w(x)}{\sum_{x \in T} w(x)} \tag{3}$$

Garciarena *et al.* [7] proposed a similar definition for the neighborhood of an ML pipeline, where they changed a randomly chosen algorithm or hyperparameter by another feasible value. The neighborhood we proposed here can be considered as an extension of their approach, and is based on the typical mutation operator used in grammar-based genetic programming [11]. For example, the proposed neighborhood can change all preprocessing steps of a pipeline at once, something the original approach does not allow.

Distance Between Pipelines: In the context of ML pipelines, the distance $dist(T_x, T_y)$ between two trees T_x and T_y must reflect the impact of changing either an algorithm or a hyperparameter. For example, the impact of changing a linear model by an ensemble for the classification task will probably have a more significant impact in fitness than changing the number of models used by the ensemble (a hyperparameter). For this reason, determining the distances between any two symbols of the grammar is not straightforward and depends on expert knowledge. In order to define these distances, we classified the symbols of the grammar into 17 disjoint sets $A_0, A_1, \ldots, A_{16}$:

- A_0: *NULL* symbol
- A_1: *<start>* symbol
- A_2: *<preprocessing>* symbol
- A_3: *<classification>* symbol
- A_4: Imputation algorithms
- A_5: Data range manipulation algorithms
- A_6: Dimensionality manipulation algorithms
- A_7: Naïve Bayes
- A_8: Linear models
- A_9: Neural networks
- A_{10}: Nearest neighbors
- A_{11}: Discriminant analysis
- A_{12}: Trees
- A_{13}: Ensembles
- A_{14}: Discrete hyperparamenters
- A_{15}: Continuous hyperparameters
- A_{16}: Categorical hyperparameters

A list of the algorithms in each set is available online (See footnote 2). The set A_0 is reserved to a special *NULL* symbol, used to treat cases in which a node in a tree does not have a corresponding node in the other. For $i, j \in \{0, 1, \ldots, 16\}$, the distance $d(x, y)$ between two symbols $x \in A_i$ and $y \in A_j$ is defined as a constant

Table 1. Distances $d(x, y)$ between symbols w.r.t. their partitions.

	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}	A_{16}
A_0	1	0	8	0	4	4	4	0	0	0	0	0	0	0	0	0	0
A_1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A_2	8	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A_3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
A_4	4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
A_5	4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
A_6	4	0	0	0	0	0	1	2	2	2	2	2	2	2	0	0	0
A_7	0	0	0	0	0	0	2	1	2	2	2	2	2	2	0	0	0
A_8	0	0	0	0	0	0	2	2	1	2	2	2	2	2	0	0	0
A_9	0	0	0	0	0	0	2	2	2	1	2	2	2	2	0	0	0
A_{10}	0	0	0	0	0	0	2	2	2	2	1	2	2	2	0	0	0
A_{11}	0	0	0	0	0	0	2	2	2	2	2	1	2	2	0	0	0
A_{12}	0	0	0	0	0	0	2	2	2	2	2	2	1	2	0	0	0
A_{13}	0	0	0	0	0	0	2	2	2	2	2	2	2	1	0	0	0
A_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0.5	0.5
A_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0.5	0.5
A_{16}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0.5	0.5

that depends on the class a symbol belongs to. If x and y have the same label, $d(x, y) = 0$. Table 1 shows the values of the constants used in this work.

Note that to make this analysis possible, the grammar defined here limits the pipelines to have at most three preprocessing algorithms and exactly one classifier. Given this restriction, we represent a tree T_i with root r_i as $T_i = r_i(c_1^{(i)}, c_2^{(i)}, \ldots, c_m^{(i)})$, where the root has m children nodes, denoted by $c_j^{(i)}$, $j \in \{1, 2, \ldots, m\}$. Each node is represented by its label (i.e., its name) and can be considered as the root of a subtree. Let us consider T_x as the pipeline in Fig. 1. r_x corresponds to the *<start>* node. It has two children, $c_1^{(x)}$ and $c_2^{(x)}$, which correspond to the nodes *<preprocessing>* and *<classification>*, respectively. We define a function $ch_1(N)$ that returns the first child of node N. In this example, the *<imputation>* group is denoted as $ch_1(c_1^{(x)})$, whereas *<bernoulliNB>* is given by $ch_1(c_2^{(x)})$.

The distance between two trees T_x and T_y will initially depend on whether they include preprocessing steps or only a classification algorithm. Equation 4 shows four possible cases: neither T_x nor T_y have preprocessing steps, and only the distance from the classification algorithm ($dist_{clf}$) is accounted for (**C1**); both trees have preprocessing steps, and we calculate the distances from the two sides of the tree ($dist_{pre}$ and $dist_{clf}$) (**C2**); only T_x (**C3**) or T_y (**C4**) have a preprocessing step, so we calculate the distance between the classification subtrees and add a constant k to the distance, where $k = d(\textit{<preprocessing>}, NULL)$ is the distance between the *<preprocessing>* non-terminal and the *NULL* symbol, which is greater than the distance between any two preprocessing algorithms.

$$dist(T_x, T_y) = \begin{cases} dist_{clf}(ch_1(c_1^{(x)}), ch_1(c_1^{(y)})) & \text{C1} \\ dist_{pre}(c_1^{(x)}, c_1^{(y)}) + dist_{clf}(ch_1(c_2^{(x)}), ch_1(c_2^{(y)})) & \text{C2} \\ k + dist_{clf}(ch_1(c_1^{(x)}), ch_1(c_2^{(y)})) & \text{C3} \\ k + dist_{clf}(ch_1(c_2^{(x)}), ch_1(c_1^{(y)})) & \text{C4} \end{cases} \tag{4}$$

To the best of our knowledge, the way we calculate the distance between two preprocessing subtrees cannot be expressed in closed form, and function $dist_{pre}$ is described in Algorithm 1, where $children(N)$ is a function that returns all the children of node N. Sets A and B are initialized with the preprocessing groups of trees T_x and T_y, respectively (line 3). The first component of the distance is calculated as the distance from all groups that are only present in one of the trees to the *NULL* symbol (lines 4 and 5). The second component consists of the distances between groups that are present in both trees (line 6). The loop in lines 7–16 compares the groups in the intersection. Function $get\_node(l, X)$ returns the node in set X whose label is l. For each group, if the algorithms in trees T_x and T_y are different, we add their distance to the total distance (line 11). If they are the same, we add to the total distance the distance between the values of their hyperparameters (lines 13 and 14). The total distance is then returned in line 17.

As an example, consider that Algorithm 1 receives the pipelines from Figs. 1 and 2b as T_x and T_y, respectively. In line 3, set A receives node *<imputation>*

and B receives nodes *<imputation>* and *<dimensionality>*. Thus, the difference between the two sets is composed of node *<dimensionality>* and its distance to the *NULL* symbol is added to the total distance. In line 4, *intersect* gets the symbol *<imputation>* and the corresponding nodes in trees T_x and T_y are retrieved in lines 8 and 9. $algA$ and $algB$ correspond to the same algorithm, so the distances between the values of their hyperparameters are added to the total distance.

Algorithm 1. Distance between preprocessing subtrees

1: **procedure** $\text{DIST}_{pre}(T_x, T_y)$ ▷ The root of T_x and T_y is the *<preprocessing>* symbol
2: $distance \leftarrow 0$
3: $A \leftarrow \{n \mid n \in children(T_x)\}$; $B \leftarrow \{n \mid n \in children(T_y)\}$
4: $diff \leftarrow (A - B) \cup (B - A)$
5: $distance \leftarrow distance + \sum_{i=1}^{|diff|} d(diff_i, NULL)$
6: $intersect \leftarrow A \cap B$
7: **for all** $label \in intersect$ **do**
8: $algA \leftarrow ch_1(get\_node(label, A))$
9: $algB \leftarrow ch_1(get\_node(label, B))$
10: **if** $algA \neq algB$ **then** ▷ Different algorithms
11: $distance \leftarrow distance + d(algA, algB)$
12: **else** ▷ Same algorithm; check hyperparameters
13: $hpA \leftarrow children(algA)$; $hpB \leftarrow children(algB)$
14: $distance \leftarrow distance + \sum_{i=2}^{|hpA|} d(ch_1(hpA_i), ch_1(hpB_i))$
15: **end if**
16: **end for**
17: **return** $distance$
18: **end procedure**

Equation 5 handles the case of the distance between the classification algorithms. In the first case of the equation, the algorithms are the same. Thus, the roots r_1 and r_2, which correspond to the non-terminals with the names of the algorithms, have the same number of children, m. In this case, the distance between the trees is the summation of the distance between the values of the hyperparameters of each algorithm. If the algorithms are the same, the distance between the trees is the distance between the algorithms, given in Table 1.

$$dist_{clf}(T_x, T_y) = \begin{cases} \sum_{j=2}^{m} d(ch_1(c_j^{(x)}), ch_1(c_j^{(y)})) & \text{if } r_x = r_y, \\ d(r_x, r_y) & \text{otherwise.} \end{cases} \quad (5)$$

3.3 Fitness Function

The final component of the fitness landscape is a fitness function $f : X \rightarrow \mathbb{R}$ that maps each element of the set X of configurations to a real number. Here we deal with multiclass classification problems with class imbalance. Therefore, we defined the fitness function as the weighted F-measure [23] to evaluate the pipeline's learning model on the dataset of interest.

F-measure is defined in Eq. 6 for binary classification problems, where TP, FP and FN are the number of true positives, false positives and false negatives of the pipeline's learning model on the dataset, respectively.

$$F\text{-measure} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}, \tag{6}$$

As we deal with multiclass classification problems, we use a one-vs-all approach, transforming a problem of c classes into c binary classification problems to calculate the F-measure. We then calculate a weighted average of the F-measure over the c binary classification problems.

4 AutoML Fitness Landscape Analysis

For continuous optimization problems with two variables, fitness landscapes can be visualized, where the XY plane is the search space and fitness represents the third dimension. However, given the complexity of real world problems, this approach is not feasible. Several metrics have been proposed to describe and compare fitness landscapes for different problems. These metrics evaluate a number of features of optimization problems that play a role in the performance of search algorithms, such as modality, fitness distribution in the search space, ruggedness, degree of variable interdependency, evolvability, neutrality, among others [14,19].

After we have defined the fitness landscape of AutoML problems, we will use these metrics to perform an analysis of the characteristics of this space. We focus on two metrics: the fitness distance correlation (FDC) and the mean neutrality ratio of the landscape. FDC is a popular way of measuring how the fitness function correlates with the distance to the global optimum, which is a way of measuring problem difficulty [9]. Neutrality, on the other hand, indicates the presence of regions in the search space with equal (or nearly equal, in the case of continuous spaces) fitness function values, which can have positive or negative impacts on the performance of optimization algorithms [14].

Fitness Distance Correlation: The fitness distance correlation (FDC) measure was proposed by the authors in [9] to give a global view of problem difficulty for genetic algorithms, but it has been frequently used as a metric to evaluate the fitness landscape of other optimization problems [14]. In its original formulation, FDC requires knowledge of the global optimum, which is unfeasible for AutoML problems. The authors in [10] proposed an adaption of FDC, called FDC_s, for continuous problems with no known global optimum. Given a sample of n points $X = \{x_1, \ldots, x_n\}$ from the search space with associated fitness values $F = \{f_1, \ldots, f_n\}$ with mean $\overline{f}$, the best point in the sample is denoted by x^*. The Euclidean distance from x^* to every point $x_i \in X$ is denoted by $D^* = \{d_1^*, \ldots, d_n^*\}$, with mean $\overline{d^*}$. FDC_s is given by Eq. 7. From here on, we denote FDC_s simply by FDC.

$$FDC_s = \frac{\sum_{i=1}^{n}(f_i - \overline{f})(d_i^* - \overline{d^*})}{\sqrt{\sum_{i=1}^{n}(f_i - \overline{f})^2}\sqrt{\sum_{i=1}^{n}(d_i^* - \overline{d^*})^2}} \tag{7}$$

FDC returns a value between -1 (perfect anti-correlation) and +1 (perfect correlation). For maximization problems, search spaces with low FDC values

Table 2. Datasets used in the experiments.

Dataset	Instances	Features	Classes	Missing	Source
breast-w	699	9	2	Yes	OpenML
diabetes	768	8	2	No	OpenML
stalog-segment	2310	19	7	No	UCI
vehicle	846	18	4	No	OpenML
wilt	4839	5	2	No	OpenML
wine-quality-red	1599	11	6	No	OpenML

are considered easy, values around 0 are difficult, and high values correspond to misleading spaces [9,10]. Given the nature of the AutoML search space, we replace the Euclidean distances D^* used in Eq. 7 by the distance measure defined in Eq. 4.

Neutrality: Neutrality identifies the presence of regions in the landscape with equal or similar fitness [17]. Its role in determining the ability of an optimization method to find good solutions has been a topic of discussion, especially in the context of evolutionary algorithms. There is evidence that neutrality can both make the search space easier to explore [21] or get some algorithms stuck in regions of the search space with equal fitness, preventing them from exploring areas with possibly better results [14].

In the context of AutoML, we define a neutral neighborhood $N^{\approx}(s)$ of a solution s as the set $N^{\approx}(s) = \{s' \in N(s) \mid |f(s') - f(s)| < \delta\}$ for some small constant $\delta \geq 0$, where $f(s)$ is the fitness of s and $N(s)$ is a sample of the complete neighborhood of s, as defined in Sect. 3.2.

The cardinality of $N^{\approx}(s)$ is called the neutrality degree of s, whereas the neutrality ratio of s is given by $|N^{\approx}(s)|/|N(s)|$ [21]. These metrics give us an overview of the neutrality level of the landscape.

5 Experimental Analysis

In this section, we present the FDC and neutrality results for six classification datasets, obtained either from UCI [2] or OpenML [22]. Table 2 summarizes their main characteristics, including the number of instances, features and classes, the presence or absence of missing values, and the data source. Correlation analyses are reported considering Spearman's rank correlation coefficient (ρ) and the two-sided p-value for the hypothesis test (null hypothesis is that the two sets of data are uncorrelated) [25].

All pipeline configurations were generated using algorithms implemented in the Python library Scikit-learn [13] and evaluated using 5-fold cross-validation. All results reported correspond to an average of 30 independent samples from the search space.

5.1 Fitness Distance Correlation Analysis

For the FDC analysis, we generated 30 random samples of the search space of varying sizes, ranging from 500 to 3,000 in intervals of 500. The pipelines in each sample were generated by randomly selecting production rules from the grammar.

Figure 3 shows the FDC values for different sample sizes. Observe that increasing the sample size has little effect on FDC, showing our sample is able to capture the overall trend of this metric in the evaluated search space. We found a slight positive correlation between FDC and the mean fitness ($\rho = 0.222$, $p < 0.01$), which is somewhat unexpected, since higher FDC values (in our case, values that are closer to 0) should be related to harder problems.

We tried to find a relation between the mean values of accuracy reported in the OpenML repository for the datasets used in the experiments and the values of FDC. For example, *breast-w* has mean accuracy of 0.93 and FDC values always smaller than -0.2. *Diabetes*, in turn, has a mean accuracy of 0.75 and values of FDC always in the interval $[-0.05, 0]$. However, this is not consistent for all datasets. *Wilt*, for instance, has an accuracy of 0.98 and FDC values as small as *diabetes*. Analysis of FDC for a greater number of datasets and looking at their main characteristics are yet to be performed. Further, FDC results are correlated with the distance measure used in this work, and a more in-depth evaluation of this measure is also subject of future work.

However, for AutoML problems, there may be other factors related to low fitness apart from the difficulty of the problem. In order to fully understand the relation between FDC and problem difficulty in the context of AutoML, further experimentation varying the search space is necessary.

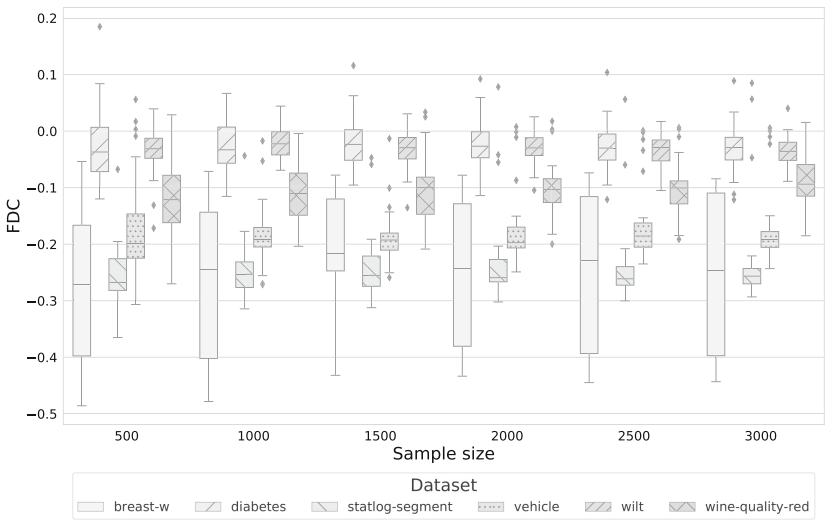


Fig. 3. FDC values for different sample sizes.

5.2 Neutrality Analysis

For the analysis of the neutrality of the search space, we performed random walks starting from a random position. For each point of the walk (solution), we evaluated the fitness function of a given number of neighbors and analyzed the neutrality ratio. One of the neighbors was then selected as the next starting point of the walk. Here we report the mean neutrality ratio of the complete walk.

Recall that, for continuous spaces, we consider regions that are neutral within a specified tolerance δ for the fitness difference between two solutions. For each dataset, we defined δ as the standard deviation of the mean fitness of 30 independent random samples of size 1000.

Figure 4 shows the neutrality ratios for different walk lengths (100, 200, 300 and 400) and neighborhood sizes (5, 10, 15 and 20). The tolerance δ for each dataset is shown in the title of the corresponding plot. As we can see, increasing the walk length and the neighborhood size does not have a strong impact on the average neutrality ratio, but the variance decreases. We found a strong positive correlation between the mean neutrality ratio and the fitness value ($\rho = 0.715$, $p = 0.0$). This result shows higher neutrality in regions of the search space with higher fitness values. However, in order to understand how neutrality affects the performance of different AutoML optimization methods, further experimentation with variations of the search space and its neutrality ratio are also necessary.

6 Related Work

Fitness landscape analysis has been vastly explored for typical optimization problems [14], but the literature regarding such analysis for machine learning problems is scarce. Much of the effort has been directed to neural network error landscapes. Rakitianskaia *et al.* [16] measured FDC, ruggedness and gradients to evaluate the error landscape of fully-connected neural networks used for classification. They showed that the ruggedness of the landscapes decreases with an increase in the number of hidden layers of the network, making the landscape "harder" to explore, whereas the results for FDC indicate that it can be used to determine the searchability of different network architectures for specific problems. Another study analyzed neutrality in such landscapes, whose presence can hinder population-based methods for training neural networks [1]. The authors proposed two measures of neutrality based on random walks on the landscape and suggested that they can be used to study the relation between neutrality and the performance of search algorithms.

In [4], the authors explored different subsets of the unbounded neural network search space using random walks. They found high-magnitude fitness gradients and more rugged landscapes for larger search spaces, specially for large steps of the random walk. Searchability metrics, on the other hand, decrease with an increase in the size of the search space. These properties reflect a greater difficulty in searching larger spaces. A subsequent study by the same group proposed a progressive random walk method for sampling network error landscapes [3]. The authors noted that methods based on random walks may not

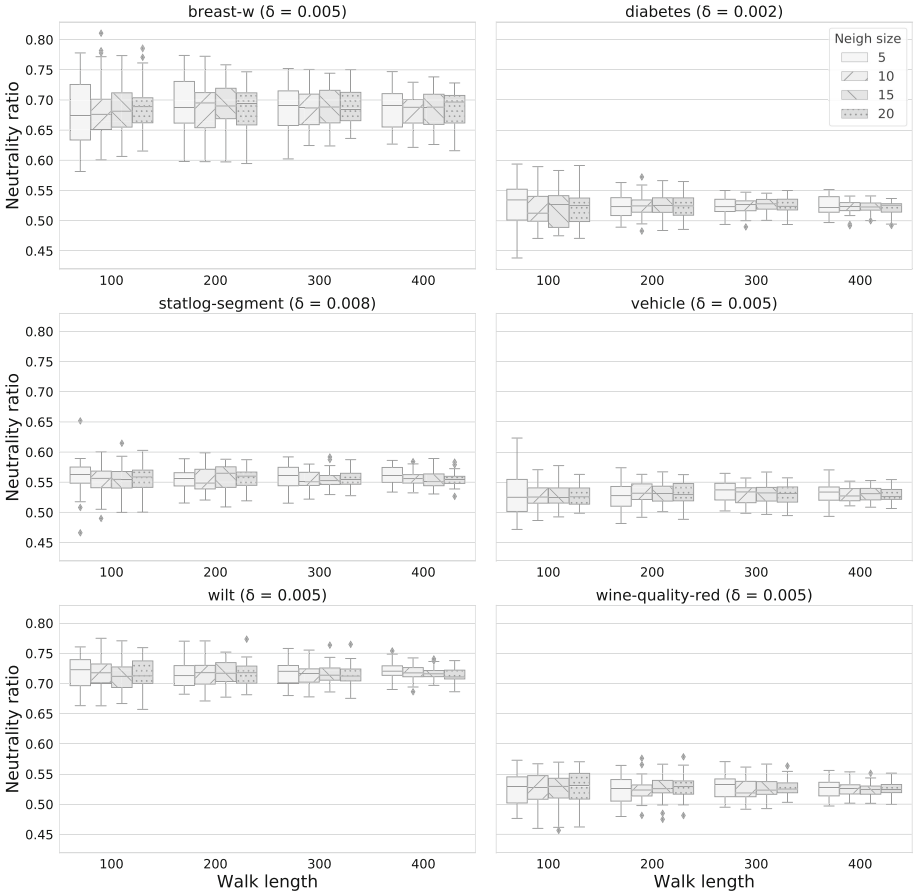


Fig. 4. Neutrality ratios for varying walk lengths and neighborhood sizes. The tolerance δ for each dataset is shown in parentheses.

cover regions with high fitness values, thus not representing the search space well. The results showed that the proposed method is more computationally efficient than population-based walks and is very successful in finding areas of high fitness.

The landscape of the algorithm configuration problem has been evaluated in [15] in terms of the modality and convexity of parameter responses. The authors defined parameter response slices by parameter p within a given window around an optima found by the Sequential Model-based Algorithm Configuration (SMAC), keeping all other parameters fixed and measuring the performance of the algorithm as a function of p. This procedure is repeated for all parameters being considered. They evaluated algorithms for three typical optimization problems, namely SAT, MIP and TSP, and concluded that many of the parameter slices appear to be uni-modal and convex, both on instance sets and on indi-

vidual instances, although the former leads to a more rugged landscape. This algorithm configuration analysis is related to the problem of hyperparameter optimization of machine learning algorithms, but it does not consider neither algorithm selection nor categorical parameters.

Garcianera *et al.* [7], in turn, performed an analysis of a subset of the search space explored by TPOT [12], an AutoML tool that uses genetic programming to evolve machine learning pipelines for regression and classification problems. The authors defined a neighborhood relation in which two pipelines are neighbors if they differ in a single algorithm or parameter. Using a reduced grid search, they compared the classification accuracy of TPOT with stochastic, random-restart hill climbing and random search. The results suggest the existence of several regions with high fitness, but which are prone to overfitting. However, the paper fails to analyze other characteristics of the fitness landscape and how they can influence the performance of optimization methods.

7 Conclusions and Future Work

The main contribution of this paper is the definition of a fitness landscape for AutoML problems. We proposed a flexible representation for machine learning pipelines that captures the relative importance of changing an algorithm by another or modifying the value of a hyperparameter. We use this representation to define a notion of neighborhood and the distance between pipelines. We found a strong correlation between the mean fitness ratio and fitness values, and a high correlation between fitness values and neutrality.

Having defined the components of the AutoML search space, the next steps include modifying the search space to evaluate how the metrics change in response to the size of the space. We also plan on testing other sampling strategy to take into account the differences in the size of the search space induced by categorical, discrete and continuous hyperparameters. Another possible direction of future work is analysing how different AutoML optimization methods behave in the presence of different levels of neutrality and for different FDC values.

Acknowledgments. The UFMG authors would like to thank FAPEMIG, CNPq and CAPES for their financial support. This work has also been partially funded by ATMOSPHERE (H2020 777154 and MCTIC/RNP 51119).

References

1. van Aardt, W.A., Bosman, A.S., Malan, K.M.: Characterising neutrality in neural network error landscapes. In: Proceedings of the Congress on Evolutionary Computation, pp. 1374–1381. IEEE (2017)
2. Asuncion, A., Newman, D.: UCI machine learning repository (2007)
3. Bosman, A.S., Engelbrecht, A.P., Helbig, M.: Progressive gradient walk for neural network fitness landscape analysis. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1473–1480. ACM (2018)

4. Bosman, A.S., Engelbrecht, A., Helbig, M.: Search space boundaries in neural network error landscape analysis. In: Proceedings of the Symposium Series on Computational Intelligence, pp. 1–8. IEEE (2016)
5. Ekárt, A., Németh, S.Z.: A metric for genetic programs and fitness sharing. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) EuroGP 2000. LNCS, vol. 1802, pp. 259–270. Springer, Heidelberg (2000). https://doi.org/10.1007/978-3-540-46239-2_19
6. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Proceedings of the Advances in Neural Information Processing Systems, pp. 2962–2970 (2015)
7. Garciarena, U., Santana, R., Mendiburu, A.: Analysis of the complexity of the automatic pipeline generation problem. In: Proceedings of the Congress on Evolutionary Computation, pp. 1–8. IEEE (2018)
8. Hutter, F., Kotthoff, L., Vanschoren, J. (eds.): Automated Machine Learning. TSSCML. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-05318-5. http://automl.org/book
9. Jones, T., Forrest, S.: Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In: Proceedings of the 6th International Conference on Genetic Algorithms, pp. 184–192 (1995)
10. Malan, K.M., Engelbrecht, A.P.: Characterising the searchability of continuous optimisation problems for PSO. Swarm Intell. **8**(4), 275–302 (2014). https://doi.org/10.1007/s11721-014-0099-x
11. Mckay, R.I., Hoai, N.X., Whigham, P.A., Shan, Y., O'Neill, M.: Grammar-based genetic programming: a survey. Genet. Program Evolvable Mach. **11**(3–4), 365–396 (2010). https://doi.org/10.1007/s10710-010-9109-y
12. Olson, R.S., Moore, J.H.: TPOT: a tree-based pipeline optimization tool for automating machine learning. In: Hutter, F., Kotthoff, L., Vanschoren, J. (eds.) Automated Machine Learning. TSSCML, pp. 151–160. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-05318-5_8
13. Pedregosa, F., et al.: Scikit-learn: machine learning in python. JMLR **12**(Oct), 2825–2830 (2011)
14. Pitzer, E., Affenzeller, M.: A comprehensive survey on fitness landscape analysis. In: Klempous, R., Suárez Araujo, C.P. (eds.) Recent Advances in Intelligent Engineering Systems, vol. 378, pp. 161–191. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-23229-9_8
15. Pushak, Y., Hoos, H.: Algorithm configuration landscapes: In: Auger, A., Fonseca, C.M., Lourenço, N., Machado, P., Paquete, L., Whitley, D. (eds.) PPSN 2018. LNCS, vol. 11102, pp. 271–283. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99259-4_22
16. Rakitianskaia, A., Bekker, E., Malan, K.M., Engelbrecht, A.: Analysis of error landscapes in multi-layered neural networks for classification. In: Proceedings of the 2016 IEEE Congress on Evolutionary Computation, pp. 5270–5277. IEEE (2016)
17. Reidys, C.M., Stadler, P.F.: Neutrality in fitness landscapes. Appl. Math. Comput. **117**(2–3), 321–350 (2001). https://doi.org/10.1016/S0096-3003(99)00166-6
18. Sipser, M.: Introduction to the Theory of Computation. 3rd edn. Cengage Learning (2012)
19. Stadler, P.F.: Fitness landscapes. In: Lässig, M., Valleriani, A. (eds.) Biological Evolution and Statistical Physics, vol. 585, pp. 183–204. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45692-9_10

20. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 847–855. ACM (2013)
21. Vanneschi, L., Pirola, Y., Mauri, G., Tomassini, M., Collard, P., Verel, S.: A study of the neutrality of boolean function landscapes in genetic programming. Theor. Comput. Sci. **425**, 34–57 (2012)
22. Vanschoren, J., Van Rijn, J.N., Bischl, B., Torgo, L.: OpenML: networked science in machine learning. ACM SIGKDD Explor. Newsl. **15**(2), 49–60 (2014)
23. Witten, I.H., Frank, E., Hall, M.A., Pal, C.J.: Data mining: practical machine learning tools and techniques, 4th edn. Morgan Kaufmann Publishers Inc., Burlington (2016)
24. Zöller, M.A., Huber, M.F.: Survey on automated machine learning. arXiv preprint arXiv:1904.12054 (2019)
25. Zwillinger, D.: CRC standard mathematical tables and formulae. Chapman and Hall/CRC, London/Boca Raton (2002)