UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Cristiano Guimarães Pimenta

Characterization of Automated Machine Learning Fitness Landscapes

Belo Horizonte
2023

Cristiano Guimarães Pimenta

Characterization of Automated Machine Learning Fitness Landscapes

Final Version

Thesis presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Gisele Lobo Pappa
Co-Advisor: Alex Guimarães Cardoso de Sá

Belo Horizonte
2023

Pimenta, Cristiano Guimarães.

P644c Characterization of automated machine learning fitness landscapes [recurso eletrônico] / Cristiano Guimarães Pimenta – 2023.

1 recurso online (99 f. il., color.) : pdf.

Orientadora: Gisele Lobo Pappa
Coorientador: Alex Guimarães Cardoso de Sá

Dissertação(Mestrado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciências da Computação.
Referências: f. 60-68

1. Computação – Teses. 2. Aprendizado do computador – Teses. 3. Otimização combinatória - Teses.4. Fitness landscape – Teses. I. Pappa, Gisele Lobo. II. Sá, Alex Guimarães Cardoso de. III. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Computação. IV.Título.

CDU 519.6*82(043)

Ficha catalográfica elaborada pela bibliotecária Irénquer Vismeg Lucas Cruz
CRB 6/819 - Universidade Federal de Minas Gerais - ICEx

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

CHARACTERIZATION OF AUTOMATED MACHINE LEARNING FITNESS LANDSCAPES

CRISTIANO GUIMARÃES PIMENTA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores(a):

Profa. Gisele Lobo Pappa - Orientadora
Departamento de Ciência da Computação - UFMG

Dr. Alex Guimarães de Sá - Coorientador
CBCI - Baker Heart and Diabetes Institute

Prof. Renato Vimieiro
Departamento de Ciência da Computação - UFMG

Prof. Ricardo Bastos Cavalcante Prudêncio
Centro de Informática - UFPE

Belo Horizonte, 21 de junho de 2023.

Documento assinado eletronicamente por **Gisele Lobo Pappa**, **Professora do Magistério Superior**, em 28/08/2023, às 13:32, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Renato Vimieiro**, **Professor do Magistério Superior**, em 30/08/2023, às 10:07, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Alex Guimarães Cardoso de Sá**, **Usuário Externo**, em 01/09/2023, às 10:49, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Ricardo Bastos Cavalcante Prudêncio**, **Usuário Externo**, em 06/09/2023, às 14:20, conforme horário oficial de Brasília, com fundamento no art. 5º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufmg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **2572461** e o código CRC **CBF23D86**.

Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, FAPEMIG, MasWeb and ATMOSPHERE (H2020 777154 and MCTIC/RNP 51119).

I would like to thank my family, friends, advisors and undergraduate research assistants Felipe Aragão Nogueira de Freitas, Isadora Salles and Paulo Fraga for their support.

"*Once I had a dream, and this isn't it.*"
(Tuomas Holopainen - Dark Chest of Wonders [adapted])

Resumo

Aprendizado de Máquina Automatizado (AutoML) tem o objetivo de selecionar e configurar *pipelines* de aprendizado de máquina automaticamente, sem exigir conhecimentos profundos do usuário. Métodos de AutoML utilizam um espaço de busca que contém possíveis soluções e tentam encontrar o melhor *pipeline* para um problema de aprendizado específico. Entretanto, pouco se sabe sobre quais são as características desses espaços de busca e como elas afetam o desempenho de métodos de busca. Uma forma de descrever os espaços de busca é por meio de Análise de *Fitness Landscape* (FLA), uma técnica muito utilizada para descrever o espaço de busca de problemas de otimização combinatória. O presente trabalho adapta métricas clássicas de FLA, tais como Neutralidade, Correlação de Distância de *Fitness* (FDC) e Distância de Correlação ao contexto de AutoML, cujos espaços de busca são complexos, uma vez que contêm variáveis discretas, contínuas, categóricas e condicionais, de forma totalmente independente do método de busca utilizado para explorar o espaço. Além disso, é feita uma avaliação de como as características do espaço de busca afetam o desempenho de dois métodos de busca baseados em otimização Bayesiana: *Tree-structured Parzen Estimator* (TPE) e *Sequential Model-based Algorithm Configuration* (SMAC). De forma a utilizar FLA no contexto de AutoML, nós propomos uma representação em árvore para os *pipelines* de aprendizado de máquina capaz de capturar sua semântica, uma definição de vizinhança baseada em um operador de mutação e uma medida semântica de distância entre *pipelines*. Análises de Neutralidade sugerem que espaços de busca maiores tendem a ter mais áreas com valores iguais, ou quase iguais, de *fitness*, uma característica que pode melhorar a habilidade do TPE de explorar o espaço e encontrar boas soluções. Espaços de busca maiores tendem a ser mais enrugados, de acordo com a métrica de Distância de Correlação, e normalmente são mais difíceis para os otimizadores. FDC se mostrou uma métrica pouco informativa em relação à dificuldade do problema de encontrar o melhor *pipeline* de aprendizado de máquina. Além disso, a utilização de ótimos locais para calcular a métrica pode levar a resultados bastante diferentes em comparação ao uso do ótimo global, cujo cálculo é normalmente inviável para problemas de AutoML. Por outro lado, desempenho do otimizador SMAC se mostrou menos afetado por alterações nas características do espaço, quando comparado ao TPE.

Palavras-chave: Análise de *Fitness Landscape*. Aprendizado de Máquina Automatizado. Espaços de Busca. Otimização.

Abstract

Automated Machine Learning (AutoML) aims at automatically selecting and configuring complete machine learning pipelines without requiring deep user expertise. AutoML methods utilize a search space of possible solutions and try to find the best pipeline for a given learning problem. However, there is little knowledge about the characteristics of such spaces and how they relate to the performance of search methods. One way of exploring them is using Fitness Landscape Analysis (FLA), a technique commonly used to describe the landscape of combinatorial optimization problems. This work adapts classic FLA measures, such as Neutrality, Fitness Distance Correlation (FDC) and Correlation Length, to the context of the complex fitness landscape generated by AutoML search spaces, which include discrete, continuous, categorical and conditional variables, regardless of the methods used to explore the search spaces. It also evaluates how the characteristics of the landscape affect the performance of two AutoML methods based on Bayesian optimization: Tree-structured Parzen Estimator (TPE) and Sequential Model-based Algorithm Configuration (SMAC). In order to use FLA in the context of AutoML, we propose a tree-based representation for machine learning pipelines that is able to capture their semantics, a neighborhood definition based on a mutation operator, and a semantic distance metric between pipelines. Neutrality analyses suggest that larger landscapes tend to have more areas of equal or nearly equal fitness values, a feature that can improve the ability of TPE to explore the search space and find good solutions. Larger search spaces tend to be more rugged, as indicated by the Correlation Length measure, and are often more challenging for the optimizers. FDC proved to be a weak measure in describing problem difficulty. Furthermore, using local optima to calculate FDC can lead to very different results when compared to using the global optimum, which is usually unfeasible to calculate for AutoML problems. On the other hand, SMAC's performance seems less affected by changes in the characteristics of the landscape.

Keywords: Fitness Landscape Analysis. Automated Machine Learning. Search Spaces. Optimization.

List of Figures

List of Tables

Contents

List of Abbreviations

- AutoML: Automated Machine Learning
- BNF: Backus-Naur Form
- BO: Bayesian Optimization
- CASH: Combined Algorithm Selection and Hyperparameter optimization
- CE: Classification Error
- CFG: Context-Free Grammar
- CNN: Convolutional Neural Network
- EI: Expected Improvement
- FDC: Fitness Distance Correlation
- FLA: Fitness Landscape Analysis
- GP: Genetic Programming
- HPO: Hyperparameter Optimization
- LON: Local Optima Network
- ML: Machine Learning
- MSE: Mean Squared Error
- NAS: Neural Architecture Search
- NN: Neural Network
- PSO: Particle Swarm Optimization
- SMAC: Sequential Model-based Algorithm Configuration
- SMBO: Sequential Model-Based Optimization
- TPE: Tree-structured Parzen Estimator

Chapter 1

Introduction

The past few decades have seen tremendous development in the field of Machine Learning (ML), which comprises methods that are able to learn from data, improve automatically with experience and predict future events [51; 52]. ML has been used in a wide variety of applications, such as image and speech recognition, self-driving vehicles, recommendation systems, predictive maintenance in the industry, genetics and credit analysis, to cite a few [20; 37; 43; 58; 94; 95].

In an ideal scenario, a machine learning task is performed by a practitioner who has deep knowledge in statistics, ML algorithms and the mathematics behind them. She obtains and treats the data, and then chooses an algorithm to solve the problem at hand. Having chosen the model, she has to choose its hyperparameters, train it and correctly evaluate the results. This process may have to be repeated several times until a solution with desirable performance is found. As a consequence, the process of solving an ML task is arduous, and human interference can lead to suboptimal solutions [95]. Furthermore, according to the Law Of Conservation of Generalization Performance for learning problems [72] and the "No Free Lunch" theorems [91; 89; 90], there is no optimal solution for all problems and all problem instances, so the process of building a machine learning pipeline must be repeated for each new application.

The desire of the industry and researchers from other areas to be able to use ML without a lot of investment gave rise to Automated Machine Learning (AutoML), which aims to automate the process of creating complete ML pipelines (i.e., a sequence of tasks to follow in order to perform data analysis) to any dataset without requiring deep user knowledge [30]. Therefore, AutoML democratizes the application of machine learning and data analysis to solve a wide range of research and real-world problems. Experienced ML practitioners and domain experts, with little or no technical background, can both benefit from AutoML. The former can automate tedious and time-consuming tasks, such as tuning hyperparameters, and focus more on data quality and the interpretation of the results, for example, while the latter becomes able to apply ML as a tool, without the need to spend a lot of time learning statistics, mathematics and algorithms.

Several optimization methods have been proposed to solve the problem of automatically selecting and configuring ML pipelines. Although each method tackles the problem

in a different way, they frequently show very similar results [3; 17; 18; 95]. 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 (i.e., data manipulation and learning algorithms, along with their hyperparameters) from previously designed ML pipelines. The optimization method is responsible for finding the best combinations of the building blocks to build the most effective solution according to a quality metric to a given dataset.

There is still little knowledge on how the characteristics of the search space affect the performance of AutoML optimizers. These search spaces are difficult to analyze, as they include discrete, continuous, categorical and conditional variables [30]. A better understanding of AutoML search spaces can help to explain the behavior of existing search methods and lead to the development of new ones, specifically designed to explore the peculiarities of such spaces [62]. One way to analyze the characteristics of the search space is through fitness landscape analysis (FLA) [76]. The fitness landscape of a problem is given by the fitness values (i.e., a measure of the quality of the solution) 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, namely continuous, discrete, categorical and conditional, performing FLA, in this case, is more challenging because the notion of neighborhood or distance function needed by FLA metrics is not straightforward [13; 14; 48; 87].

1.1 Objectives

The main goal of this thesis is to describe and analyze the characteristics of AutoML search spaces and how these characteristics affect the performance of search methods.

Research Question 1 (Characteristics of the search space) *What are the characteristics of the search space of algorithms $\mathcal{A}$ and the hyperparameters $\lambda^{(j)}$ of each $A_j \in \mathcal{A}$?*

Research Question 2 (Optimizer performance) *How do the characteristics of the fitness landscape affect the performance of TPE and SMAC?*

1.2 Contributions

The first contribution of this work is the definition of the components of a fitness landscape in the context of AutoML problems, namely:

- A tree-based representation of ML pipelines that is able to capture the semantics of the pipeline;
- A definition of the probabilistic neighborhood for ML pipelines;
- A distance metric between pipelines that considers their semantics.

Furthermore, our previously published work [61] was the first to apply typical fitness landscape metrics to analyze AutoML landscapes. We also give some initial insights into how the characteristics of the search space impact the performance of AutoML optimization algorithms.

1.3 Thesis organization

This thesis is organized as follows. Chapter 2 provides the formal definition of AutoML, presents the most well-known optimizers, defines Fitness Landscape Analysis and presents works dealing with fitness landscape analysis of combinatorial optimization problems, neural networks, hyperparameter optimization, algorithm selection and AutoML. Chapter 3 describes fitness landscape analysis in the context of AutoML, while Chapter 4 presents the experimental evaluation and results. Finally, Chapter 5 concludes the thesis and presents possible directions for future research.

Chapter 2

Background and Related Work

This chapter formally defines the problem of AutoML and discusses the main metrics used for fitness landscape analysis. To conclude, it discusses related works that use FLA to understand problem difficulty, including AutoML problems.

2.1 Problem definition

Before the area of AutoML was established, the problem of selecting the best algorithm and its hyperparameters was already well-studied in the literature, both in the context of machine learning and optimization problems [59].

For a given computational problem, including machine learning problems, there may exist several different algorithms, using different approaches, which try to solve it [39]. In some specific scenarios, one algorithm can always yield better results than the others. In most cases, however, no single algorithm outperforms all the others in all instances of the problem, a phenomenon known as *performance complementarity* [38]. Choosing the best algorithm for an instance of a computational problem is known as the algorithm selection problem [69].

Given a set $\mathcal{A} = \{A^{(1)}, A^{(2)}, \ldots, A^{(n)}\}$ of algorithms, the algorithm selection problem aims to find the algorithm $A^* \in \mathcal{A}$ with the best generalization performance. Given a dataset $\mathcal{D}$ and a gain function $\mathcal{G}$, algorithm selection in the context of machine learning problems is given by Equation 2.1.

$$A^* \in \underset{A \in \mathcal{A}}{\operatorname{argmax}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{G}(A, \mathcal{D}^{(i)}_{\text{train}}, \mathcal{D}^{(i)}_{\text{valid}}), \tag{2.1}$$

where $\mathcal{G}(A, \mathcal{D}^{(i)}_{\text{train}}, \mathcal{D}^{(i)}_{\text{valid}})$ is the gain achieved when A 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 [78].

However, even if the problem of selecting an algorithm is solved, the selected algorithm may still have hyperparameters. Automatically setting these hyperparameters is one of the most basic tasks in AutoML [30]. Hyperparameter optimization (HPO) methods must deal with different types of hyperparameters, namely discrete, continuous, categorical and conditional (i.e. parameters that need to be tuned depending on the value of another parameter) [30; 93].

Let A denote a machine learning algorithm and Λ be the set of all possible combinations $\lambda \in \Lambda$ of hyperparameters for A. When algorithm A is tuned with hyperparameters λ, it is denoted by A_λ. Given a dataset $\mathcal{D}$ and a gain function $\mathcal{G}$, which measures the quality of the solution, the definition of hyperparameter optimization, whose goal is to find the best combination of hyperparameter values for the task at hand, is given by Equation 2.2.

$$\lambda^* \in \underset{\lambda \in \Lambda}{\operatorname{argmax}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{G}(A_\lambda, \mathcal{D}_{\text{train}}^{(i)}, \mathcal{D}_{\text{valid}}^{(i)}), \tag{2.2}$$

where $\mathcal{G}(A_\lambda, \mathcal{D}_{\text{train}}^{(i)}, \mathcal{D}_{\text{valid}}^{(i)})$ is the gain achieved when A_λ is trained and validated on disjoint training and validation sets $\mathcal{D}_{\text{train}}^{(i)}$ and $\mathcal{D}_{\text{valid}}^{(i)}$, respectively, on each partition $1 \leq i \leq k$ of a k-fold cross-validation procedure [78].

If we consider algorithms as categorical hyperparameters of a machine learning pipeline, AutoML can be formalized as a Combined Algorithm Selection and Hyperparameter optimization (CASH) problem. Given a set $\mathcal{A} = \{A^{(1)}, A^{(2)}, \ldots, A^{(n)}\}$ of algorithms, where each algorithm $A^{(i)}$ has a hyperparameter space $\Lambda^{(i)}$, CASH aims to find the best parameterized algorithm $A_{\lambda^*}^*$ [23; 78]. In its original formulation, CASH is a minimization problem. However, given the nature of classification algorithms, dealt with in this work, and the fitness function used to evaluate them, we cast CASH as a maximization problem, replacing the loss function with a gain function. Thus, our modified version of CASH is given by Equation 2.3.

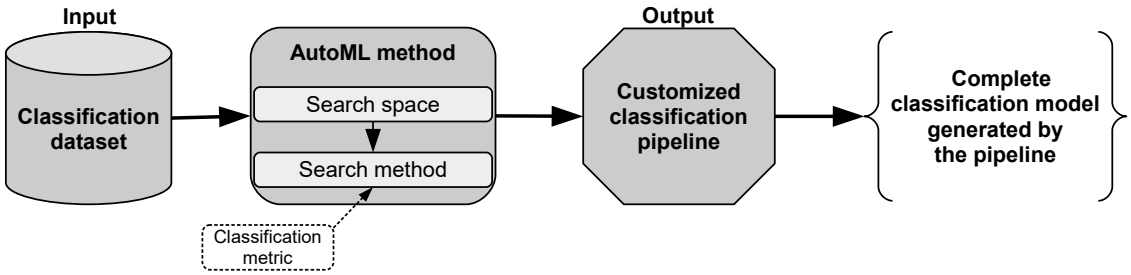
$$A_{\lambda^*}^* \in \underset{A^{(i)} \in \mathcal{A}, \lambda \in \Lambda^{(i)}}{\operatorname{argmax}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{G}(A_\lambda^{(i)}, \mathcal{D}_{\text{train}}^{(i)}, \mathcal{D}_{\text{valid}}^{(i)}), \tag{2.3}$$

The first objective of the current work (Research Question 1) is to analyze the characteristics of the search space defined by $\mathcal{A}$ and Λ, regardless of the method employed to solve the problem.

Figure 2.1 illustrates a basic framework followed by AutoML methods in order to create complete machine learning pipelines for classification problems. The AutoML method receives as input a classification dataset, composed of a set of features and a class label. The AutoML method itself has two main components: a search space, which defines the building blocks of all possible classification pipelines (i.e., the preprocessing algorithms and the classifier, along with their hyperparameters, and some optional post-

processing steps), and a search method, used to explore the search space. The output of the AutoML method is a classification pipeline customized to the input dataset, according to a classification metric, also called the fitness function. Finally, the customized pipeline generates a complete classification model, including preprocessing and postprocessing steps. Henceforth, we consider the classification pipeline as the final result of the AutoML method.

Figure 2.1: Simple AutoML framework for classification problems.



Source: adapted from [18].

Machine Learning pipelines: An ML pipeline usually consists of optional preprocessing steps (e.g., data cleaning, feature selection, and dimensionality reduction), a machine learning model (e.g., a classifier or a regressor), and some optional postprocessing steps which may be used to combine the results of several learning models, as done in Witten et al. [88].

Several methods have been proposed to solve the problem of choosing the best hyperparameter configuration or even generating complete machine learning pipelines [30; 37; 94; 95]. Such methods are usually based on Bayesian optimization (BO), evolutionary search, multi-fidelity optimization, reinforcement learning, hierarchical planning, local search and random search, among others. In this section, we focus on works from the literature that use BO, evolutionary search and multi-fidelity optimization, which are the most frequently used.

BO is a powerful framework for optimizing expensive black-box functions. The framework consists of two main components: a probabilistic surrogate model, which tries to approximate an expensive black-box function (e.g. a fitness function), and an acquisition function, which chooses the next point to be evaluated [30; 73; 75]. Bayesian optimization can be formalized as Sequential Model-Based Optimization (SMBO), a framework that takes turns fitting models and choosing other configurations to evaluate [29]. SMBO is used in methods such as TPE and SMAC, which are the basis of AutoML frameworks such as Hyperopt [5], Auto-WEKA [78] and AutoSKLearn [23], for example. TPE and SMAC will be detailed later in Section 3.5.

Another popular approach is to employ population-based methods to evolve ML pipelines and is used in AutoML frameworks such as TPOT, and RECIPE. TPOT [57] uses

tree-based genetic programming to perform a global search on the search space, whereas RECIPE [17] uses grammar-based genetic programming to overcome the common problem of generating invalid pipelines.

Other types of methods include those based on multi-fidelity optimization, such as the one proposed by Li et al. [41]. Hyperband uses a multi-armed bandit approach that focuses on making random search more efficient by adaptively allocating computational resources and performing early-stopping. Autoband [16] extends Hyperband to address the problem of algorithm configuration. BOHB [22] combines Bayesian optimization and Hyperband in order to achieve strong anytime performance and fast convergence to optimal algorithm configurations.

2.2 Fitness Landscape Analysis

Among the previously proposed optimization methods introduced to solve the problem of automatically selecting and configuring machine learning pipelines, there is not a single one that seems to outperform all the others [95]. This is in agreement with the "No Free Lunch" theorem for search and optimization, which states that no single algorithm is better than the others in all problem instances [91; 90]. However, on specific problem instances, there may be an algorithm (or a complete machine learning pipeline) that outperforms all the others [38]. Finding such a pipeline is the main goal of AutoML. Thus, knowledge about the characteristics of the search space can help to explain the performance of the algorithms or even lead to the development of better algorithms for a given problem class [45; 50; 62].

One way to analyze the characteristics of the search space is through Fitness Landscape Analysis (FLA) [62; 45; 44]. The concept of fitness landscapes was first introduced by Wright [92], in his seminal paper on genetic evolution. Although the definition lacked mathematical formalism, it served as inspiration for the development of techniques to explain other complex systems, such as heuristic search methods for combinatorial optimization problems. According to Stadler [76], a fitness landscape has three main components, described below. Changing any of the components generates a different landscape.

1. A set X of configurations (in our case, sets of algorithms and hyperparameters of AutoML pipelines, formally defined in Section 3.1);

2. A notion $\mathcal{X}$ of neighborhood or distance on X, and

3. A fitness function $f : X \rightarrow \mathbb{R}$, which corresponds to the function being optimized (e.g., classification accuracy, classification error or regression error).

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 for the same set of configurations [45; 62].

For very simple problems, fitness landscapes can be represented graphically and compared visually. However, given the complexity of real-world problems, this approach is usually not feasible [11; 19; 31]. Hence, several metrics have been proposed to describe and compare fitness landscapes for different problems or neighborhood functions. These metrics capture a number of features of optimization problems that may 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, number and size of basins of attraction and the correlation between fitness values and the distance to the global optimum, among others [45; 62; 76].

In the present work, we focus on three FLA metrics, chosen according to their popularity and the possibility to be adapted to the AutoML context: Neutrality, Correlation Length and Fitness Distance Correlation (FDC). The following sections describe each of these metrics.

2.2.1 Neutrality

One of the most widely used FLA metrics is neutrality [2; 26; 45; 53; 85]. Neutrality occurs when neighboring points on the landscape are neutral, i.e., they have the same fitness value [45; 68; 82]. Formally, given a neighborhood $N(S)$ of a solution S, the neutral neighborhood $\mathcal{N}(S)$ of S is given by Equation 2.4.

$$\mathcal{N}(S) = \{S' \in N(S) \mid f(S') = f(S)\} \tag{2.4}$$

The cardinality of $\mathcal{N}(S)$ is called the neutrality degree of solution S, whereas the neutrality ratio of S is given by $|\mathcal{N}(S)|/|N(S)|$. The overall neutrality ratio of the landscape is then given by the mean of the neutrality ratios of all possible solutions. In this work, all references to neutrality consider the overall neutrality ratio.

2.2.2 Correlation Length

Another feature of fitness landscapes that can greatly impact the performance of search algorithms is ruggedness, which can be described as the number and distribution of local optima [2; 45; 76]. Weinberger [86] proposed two metrics to quantify the ruggedness of a fitness landscape: Auto-correlation Function and Correlation Length. Auto-correlation Function describes the correlation between fitness values of sequential points of a random walk over the landscape, whereas Correlation Length measures the distance after which points on the landscape become uncorrelated. The smaller the distance, the more rugged the landscape.

Consider a random walk, described in Section 3.4.1, of length l over the landscape with fitness values $F = \{f_1, \ldots, f_l\}$. Each $f_t \in F$ is the fitness of the solution visited at time t. The Auto-correlation Function $\rho(s)$ of the landscape during this random walk with step size s, as defined by Czech [15], is given by Equation 2.5:

$$\rho(s) = \frac{\sum_{t=1}^{l-s}(f_t - \overline{f})(f_{t+s} - \overline{f})}{\sum_{t=1}^{l}(f_t - \overline{f})^2} \tag{2.5}$$

where $\overline{f}$ is the mean of F.

Given this function, the Correlation Length l is a single value inversely proportional to the ruggedness of the landscape [86] and is given by Equation 2.6:

$$\ell = -\frac{1}{\ln|\rho(1)|} \tag{2.6}$$

2.2.3 Fitness Distance Correlation

The last FLA metric discussed in this work is Fitness Distance Correlation (FDC). FDC was proposed in 1995 by Jones and Forrest [35] to give a global view of problem difficulty for genetic algorithms, but has since been widely used to evaluate the fitness landscape of various kinds of optimization problems [24; 42; 46; 53; 55].

In its original formulation, FDC measures the correlation between the fitness values of the solutions and the distance to a global optimum. For maximization problems, a correlation of -1 indicates that fitness increases as the distance to a global optimum decreases, as one would hope. Given a sample of size n of the landscape with fitness values $F = \{f_1, \ldots, f_n\}$ and a set $D = \{d_1, \ldots, d_n\}$, where each $d_i \in D$ corresponds to

the distance from solution i to the global optimum, the formal definition of FDC is given by Equation 2.7.

$$FDC = \frac{1/n \sum_{i=1}^{n}(f_i - \overline{f})(d_i - \overline{d})}{\sigma_F \sigma_D}, \tag{2.7}$$

where σ_F and σ_D are the standard deviations and $\overline{f}$ and $\overline{d}$ are the means of F and D, respectively.

2.2.4 Summary

Table 2.1 presents a summary of the FLA metrics used in this work, the characteristic of the fitness landscapes to which they relate and their interpretations.

Table 2.1: Summary of FLA metrics and fitness landscape characteristics.

Metric	**Characteristic**	**Interpretation**
Neutrality Ratio	Neutrality	Higher ratios indicate the presence of larger areas of equal fitness. Lower ratios indicate smaller areas.
Correlation Length	Ruggedness	Smaller values indicate more rugged landscapes, whereas larger values indicate smoother landscapes.
FDC	Deception with respect to local search.	For maximization problems, values close to -1 indicate easy landscapes, values around 0 are hard and values close to 1 are misleading.

2.3 Related Work

Since its introduction in 1932 [92], fitness landscape analysis went from being a highly theoretical idea in evolutionary computing to being applied to a wide range of optimization problems and machine learning [45; 44]. This chapter presents some works from the literature that use fitness landscape analysis with the goal of understanding problem difficulty. We begin by presenting works that studied the metrics described in Section 2.2 for classic optimization problems and neural networks. We then turn our focus

to works that present an overall evaluation of the complexity of AutoML-related problems, such as algorithm selection, hyperparameter optimization and pipeline generation.

2.3.1 Optimization Problems

The first applications of FLA were in the field of evolutionary computing. In his doctoral dissertation, Jones [34] presented an in-depth study about FDC for evolutionary algorithms. He showed that the metric correctly identified the difficulty of several well-known optimization problems, including problems for which the behavior of genetic algorithms surprised the researchers when they were first proposed, such as the Tanese and the royal roads functions. In some cases, although the algorithms may require more resources to solve more complicated problem instances, FDC does not show an increase in problem difficulty, which can be counterintuitive. However, as an advocate of FDC, the author argues that, even when the metric does not correlate well with problem difficulty, it is very useful, because the correlation between fitness values is a necessary, but not sufficient, condition for the landscape to be easily searchable. He concludes that FDC should only be used as an indication of the expected difficulty of the problem. Vanneschi et al. [82, 83] studied the neutrality of the landscapes of genetic programming (GP) for the even parity and multiplexer problems. They found that problem difficulty depends on the set of logical operators used by the GP. For neutral networks induced by the NAND operator, higher neutrality levels are associated with lower fitness values and the GP is unlikely to improve the quality of the solution by mutating individuals from regions with good fitness, which is not observed for networks induced by the XOR and NOT operators, for the even parity problem, or the IF operator, for the multiplexer problem. Furthermore, they conclude the XOR, NOT, and IF operators generate more neutral landscapes, which were easier for the GP, meaning that neutrality can be useful, especially when located in good regions of the fitness landscape.

Another largely studied method is particle swarm optimization (PSO). Malan and Engelbrecht [46] applied PSO to several benchmark problems and used FDC to evaluate the ability of the search method to find solutions with increasing fitness on a particular problem. The results for FDC and other searchability measures suggested that no single measure can be used to predict algorithm performance, but they should be used together. A studied performed by Harrison et al. [27] showed that FDC is highly dependent on problem instances, showing great variations between different cases of the same problem. The authors also concluded that larger search spaces tend to be more searchable, according to FDC, and that many parameter configuration landscapes, although unimodal, are not

necessarily easy to optimize. Engelbrecht et al. [21] analyzed how FDC correlates to PSO performance on several minimization benchmark problems. They showed that PSO tends to transition to an exploitation strategy faster for landscapes with higher FDC values[1], while it is slower to converge for lower FDC values, which indicate deceptive landscapes. Furthermore, they showed that PSO tends to converge more slowly for more neutral landscapes, but this effect was not significant. A recent work also used FDC to measure the relationship between the fitness of a PSO solution and the optimal values [42]. The authors proposed a method that uses FDC to balance the exploration and exploitation abilities of the PSO algorithm.

Ochoa et al. [55] studied FDC, neutrality and correlation length to understand the landscape for graph-based constructive hyper-heuristics for the timetabling problem, which is a minimization problem. All metrics were calculated for the heuristics search space, and not for the algorithm configuration space. They found moderate to high FDC values, suggesting an easy and globally convex landscape, with several wide plateaus with the same fitness. The correlation length was shown to be highly dependent on problem instances, with half the instances showing very rugged landscapes. The smoother landscapes also showed higher levels of neutrality. Ochoa and Veerapen [54] evaluated the neutrality of the landscapes of the iterated local search algorithm for the traveling salesperson problem. Using local optima networks, the authors found high levels of neutrality, but a thorough investigation of the impacts of neutrality was left for future work.

2.3.2 Artificial Neural Networks

More recently, fitness landscape analysis became a popular way of characterizing neural network error landscapes and neural architecture search, a branch of AutoML, spaces. Bosman et al. [8] studied different boundaries, i.e. different subsets, of the search space of neural network architectures. FDC analysis showed that larger search spaces are less searchable. The authors also demonstrated that more rugged regions of the landscape are harder for search methods and regions identified as more searchable, according to FDC, may be flatter. Given the sensitivity of FLA metrics to the bounds applied, the result may not hold for the complete, unbounded space.

Rakitianskaia et al. [67] evaluated error landscapes of multi-layered neural networks (NNs) for classification, using classification error (CE) and mean squared error (MSE) as error functions. For MSE, FDC indicated easily searchable landscapes. When using CE,

[1]For minimization problems, high FDC values correspond to easy landscapes. This is in contrast to the intuition given in Section 2.2.3, which considers maximization problems.

on the other hand, FDC found less searchable spaces, being close to 0 or even negative, indicating deceptive landscapes, for all problems considered. They conclude that using CE to guide the search may be inefficient, due to the fact that the landscape generated by the metric contains less useful information, and that FDC could be used to choose the most promising NN architectures for a particular problem. van Aardt et al. [81] also studied MSE error landscapes for NNs for classification problems. Using progressive random walk samples of the landscape, the authors found that regions around the origin of the MSE search space are usually more neutral when using a symmetric activation function. Another study proposed bounded and unbounded progressive gradient walks to sample NN error landscapes and find regions with good fitness [10]. Unbounded gradient walks found areas of increased neutrality, a property of the landscape that was not captured by other walks. However, these studies did not evaluate how neutrality affects the performance of search algorithms.

Bosman et al. [9] evaluated error landscapes of weight-elimination of NNs. FDC results indicated that the application of a penalty term simplifies the landscapes. However, these results proved to be misleading, because flatter landscapes lacked sufficient gradient information for the NN training algorithm, back-propagation, to perform well. The authors suggest that FDC may offer a global perspective on problem difficulty that is not suitable for neural network error landscapes.

Some studies use fitness landscape analysis to evaluate neural architecture search (NAS) search spaces, which is somewhat related to the focus of this dissertation, AutoML. Nunes et al. [53] applied FLA measures to characterize the search space of NAS methods for graph neural networks. The authors did not find high levels of neutrality, but the correct interpretation of the results requires further investigation. FDC results indicated that the problem is easy for search algorithms, a similar result to a study that evaluated NAS spaces of convolutional neural networks (CNNs) for image classification [71]. Traoré et al. [79] also studied CNNs for image classification. The correlation between fitness and distance to the optimum was visually evaluated. During the first epochs of the NAS algorithm training, they observed a positive correlation, indicating an easily searchable landscape. In later epochs, the correlation decreased, indicating the presence of a plateau.

2.3.3 Hyperparameter Optimization, Algorithm Selection and AutoML

In the previous sections, we presented works that studied the search spaces of optimization problems and neural network architectures, which are not directly related to the

problem of generating classic machine learning pipelines. The first study to address the question of whether the automatic pipeline generation problem has some key characteristics to be a reasonable research topic, such as the difference between the best result and the average solution performance, the ability to find a small set of very good solutions and the distribution of such solutions, for example, was done by Garciarena et al. [25]. They used a subset of the search space considered by TPOT [57] and limited the pipeline length to two algorithms (i.e., a preprocessor and a classifier or two stacked classifiers). The authors compared TPOT, an optimizer that uses genetic programming as a search method, to random search and a hill climbing algorithm. The results indicate there are several regions of the search space containing good solutions, so a simple random search could be enough. Additionally, the three search methods found many pipelines that performed well in the training set, but failed to generalize to unseen data, although TPOT was less affected by this problem than the other methods. Although this work did not evaluate typical fitness landscape measures, it provided the basis for the tree-based pipeline representation and neighborhood definition used in this work, which will be described in Chapter 3. It also gave initial insight regarding the complexity of the AutoML task.

Pushak [63] and Pushak and Hoos [64] claim to have been the first to investigate the properties of algorithm configuration landscapes. They evaluated the results of several optimization problem solvers by varying individual numerical hyperparameters while fixing the other hyperparameters at optimized values. The results indicate that most parameters appear to have uni-modal and convex responses, although this may not be the case for individual problem instances. FDC analyses support the hypothesis that aggregate responses on problem sets are less rugged than the response on individual instances. However, manual inspection revealed rugged areas that were not captured by FDC. A follow-up work by the same authors builds on the evaluation of one-dimensional hyperparameter slices in order to study multidimensional landscapes applied to AutoML loss landscapes [65]. They showed that most multidimensional landscapes are even more uni-modal than one and two-dimensional ones, but this must be due to hyperparameter interactions. Using a simple optimization method that tunes each hyperparameter independently, they showed that hyperparameter interactions do not greatly increase problem complexity. They rejected convexity for all evaluated landscapes, but showed that most hyperparameters yield convex responses when all other hyperparameters are fixed to their optimal values. Finally, the authors demonstrated the FDC fails to reveal the simplicity of most AutoML loss landscapes. Pushak and Hoos [66] proposed Golden Parameter Search, an automated algorithm configurator that exploits the characteristics of the landscape and is able to find the best configurations in a fraction of the time required by previous methods.

Traoré et al. [80] studied HPO in the context of AutoML to understand whether the process used to evaluate individual configurations can affect the HPO landscape. Their

results indicated the presence of large groups of solutions with the same low fitness, which can be explained by the use of accuracy as the fitness metric, resulting in several majority class predictors. Ruggedness analyses suggested that the problems are hard for local search algorithms. FDC results showed no clear correlation between fitness and distance to the global optimum, indicating hard landscapes.

Teixeira and Pappa [77] evaluated AutoML search spaces using local optima networks (LONs). LONs are directed graphs, in which the nodes represent local optima and the edges are the possible weighted transitions between them [56]. Varying the size of the neighborhood of each local optimum, the authors showed that larger neighborhoods result in less rugged landscapes, which are usually easier to search. This result suggests that, given the common desire of minimizing resource consumption, the problem seems inherently difficult. However, even with big budgets, it was shown that increasing the size of the neighborhood does not always make the landscape more searchable, due to different datasets having different characteristics. They also showed that applying a mutation operation several times and then performing a local search can help search methods to escape from a basin of attraction.

To the best of our knowledge, our previously published work was the first to apply classic fitness landscape analysis to AutoML search spaces. We also proposed a tree-based representation that is able to capture the semantics of machine learning pipelines and a measure of the distance between pipelines that exploits such semantics [61].

Chapter 3

Fitness Landscape Analysis for Automated Machine Learning

In this chapter, we discuss how the three components of a fitness landscape (namely the set of configurations, a notion of neighborhood or distance between configurations, and a fitness function [76]) can be defined in the context of AutoML. We also describe some adaptations that must be made to typical fitness landscape analysis metrics in order to apply them to the complex AutoML search spaces. Finally, we review two AutoML optimizers, which will be analyzed from the point of view of FLA.

3.1 Search Spaces

The first component of a fitness landscape is a set X of configurations, also called the search space. In the context of Automated Machine Learning, this set corresponds to all valid machine learning pipelines that can be used to solve a given learning problem. A machine learning pipeline can be defined as the sequence of methods that receive a dataset (i.e., an input vector) and produces a model of the response variable (ou variables).

For the sake of simplicity, here we only consider classification pipelines composed of up to three preprocessing steps (and an additional imputation algorithm, when necessary) and one classifier, without any postprocessing steps. The pipelines are generated by creating derivation trees by following the production rules of context-free grammars (CFGs).

Formally, a grammar G is represented by a four-tuple *<NT, T, PR, S>*, where NT represents a set of non-terminals, T a set of terminals, PR a set of production rules and S (a member of NT) the start symbol [74]. We will use a simplified version of the Backus-Naur Form (BNF) to represent grammars. This means that each production rule has, for instance, the following form: *<Start>::= <A><B> | <C> d*. This notation means that a "*<Start>*" symbol can be translated to either "*<A><B>*" or "*<C> d*".

Symbols wrapped in "<>" represent non-terminals, whereas terminals (such as d) are not bounded by "<>". The special symbol "|" represents a choice. Additionally, the symbol "#" represents a comment in the grammar (i.e., it is ignored by the grammar parser). The choice of one among all elements connected by "|" is made using a uniform probability distribution.

In the context of AutoML, the set of non-terminals defines the overall structure of the ML pipeline (i.e., the number and types of preprocessing algorithms and the type of classification algorithm), the names of the algorithms themselves and the names of their hyperparameters. The terminals define the names of the algorithms in a format that is understood by the underlying implementation and the values of the hyperparameters.

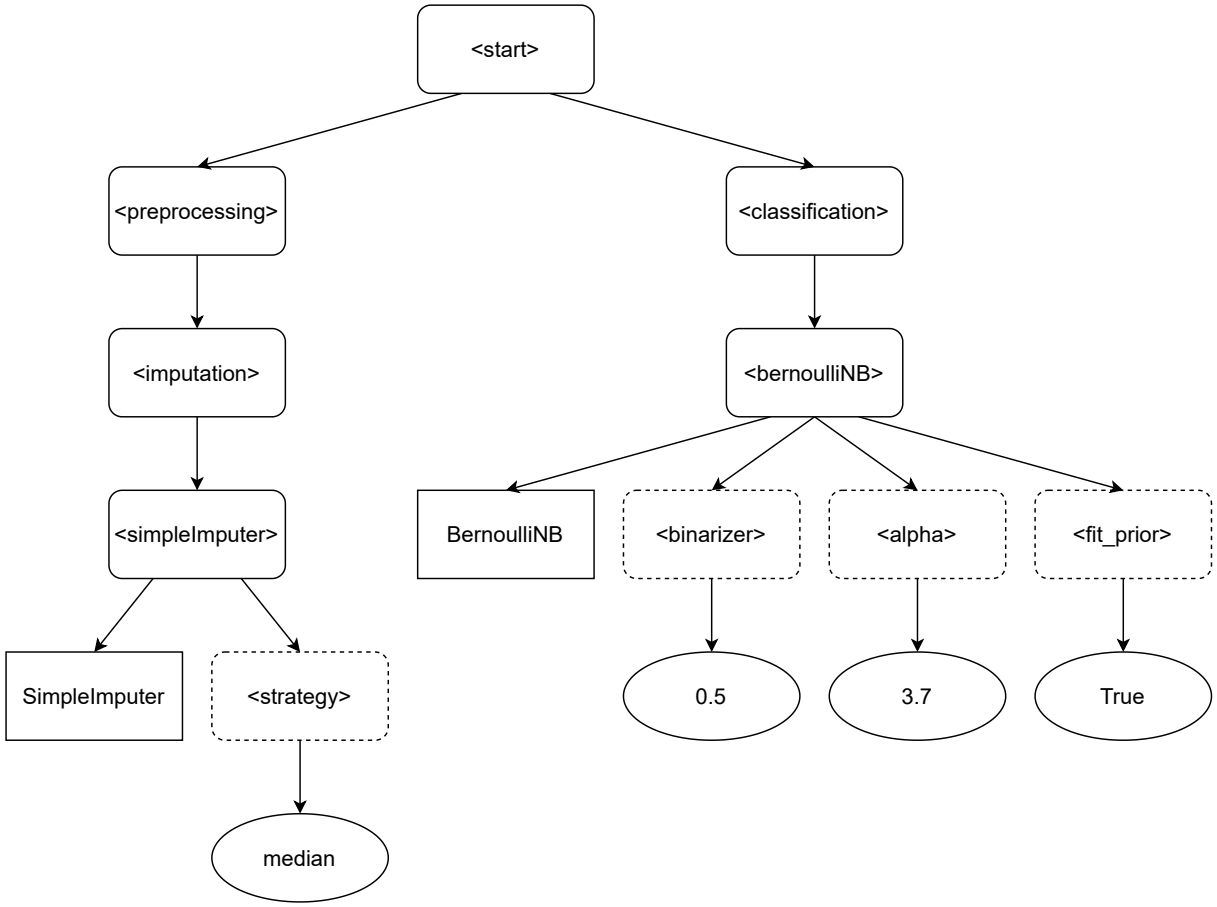
One of the benefits of using CFGs to represent the AutoML search spaces is that they organize prior knowledge (from specialists) about the problem, properly guiding the optimization process. In addition, the grammar also gives flexibility in the definition of the search space, as the grammar rules can be modified when necessary. 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. For datasets with missing values, it also guarantees that pipelines always begin with an imputation step. We propose three search spaces of different sizes and characteristics (`small`, `medium` and `large`), as well as a search space from the literature (`auto-sklearn` [23]), described below. The corresponding CFGs are listed in Appendix A.

An example of a pipeline, derived from the grammar that defines the search space `large`, described below, is shown in Figure 3.1. In the figure, algorithm names correspond to tree nodes with sharp edges; hyperparameter names are represented as rounded rectangles with dashed lines, and their values correspond to the ellipses. Pipelines are initialized at random by uniformly choosing production rules from the grammar.

In this work, we explore four different search spaces of different sizes. In order to make search space size estimations feasible, we made the simplification of always considering 100 values for continuous hyperparameters, regardless of the lower and upper bounds of the distribution. Table 3.1 summarizes their main characteristics, which will be presented in detail in the following sections. Figure 3.2 represents the search spaces as Venn diagrams. Figure 3.2a shows algorithms, whereas Figure 3.2b show hyperparameter distributions. Since search space `small` does not change hyperparameters, it is not represented.

Figure 3.1: Tree representation of a machine learning pipeline.



Source: author.

Table 3.1: Search space sizes, where $feat$ is the number of features in the dataset.

Search space	**# preprocessors**	**# classifiers**	**Size**
small	26	25	4.22e4
medium	4	5	$4.70e4 \times feat - 1.34e4$
large	26	25	$4.14e34 \times feat - 3.80e34$
auto-sklearn	18	14	$1.34e24 \times feat + 6.47e23$

3.1.1 Search space `small`

The search space `small` (Appendix A.1) is composed of 26 preprocessing algorithms and 25 classifiers. All hyperparameters are set to their default values, generating a search space with an estimated size of 4.22e4 possible combinations. For datasets with missing values, we added a mandatory imputation algorithm, which does not alter the number of combinations, considering that the hyperparameter values are fixed.

3.1.2 Search space `medium`

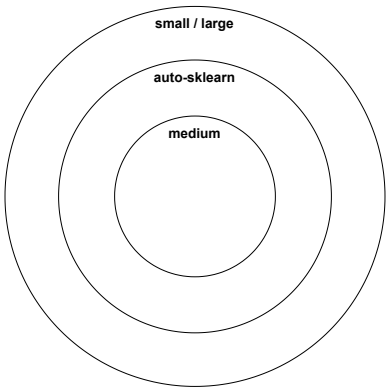
In contrast with `small`, search space `medium` (Appendix A.2) has only four preprocessing algorithms and five classifiers. However, hyperparameters are also tuned, resulting in a search space of size $4.70e4 \times feat - 1.34e4$, where $feat$ is the number of features in the dataset. The number of features is a hyperparameter of feature selection algorithms and is used to define an upper bound to the number of features to be kept. There is an alternative version for datasets with missing values, containing an additional imputation algorithm with three different values for its sole hyperparameter, which results in a threefold increase in the size of the search space.

3.1.3 Search space `large`

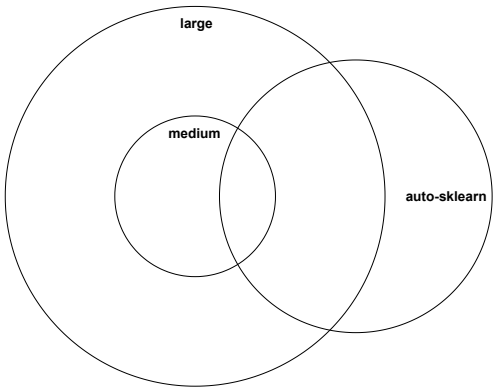
Search space `large` (Appendix A.3) contains the same algorithms (i.e., processing algorithms and classifiers) as `small`, but hyperparameters are also tuned. The distributions from which the values of the hyperparameters are selected are much broader than those used for search space `medium`, resulting in a search space with an estimated size of $4.14e34 \times feat - 3.80e34$ pipeline configurations. We also defined a version of this search space for datasets with missing values, which is three times larger than the original search space.

Figure 3.2: Venn diagrams of the search spaces (circle sizes do not reflect the size of the sets).

(a) Preprocessing and classification algorithms.



(b) Hyperparameter distributions.



Source: author.

3.1.4 Search space `auto-sklearn`

The last search space (Appendix A.4) is based on the one used by auto-sklearn [23], an AutoML system based on scikit-learn [60]. It comprises 18 preprocessing algorithms and 14 classifiers, resulting in $1.34e24 \times feat + 6.47e23$ possible configurations. Search space `auto-sklearn` always generates pipelines with an imputation algorithm, regardless of the presence of missing values in the data.

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. Considering the complexity of the AutoML search space, which contains categorical, discrete, continuous, and conditional hyperparameters, and the lack of comprehensive literature on the analysis of such spaces, we propose a simple but effective neighborhood definition and a distance metric between our tree-based pipelines.

The neighborhood definition and the distance metric are two of the main contributions of this work and were presented at the *20th European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP 2020)* [61].

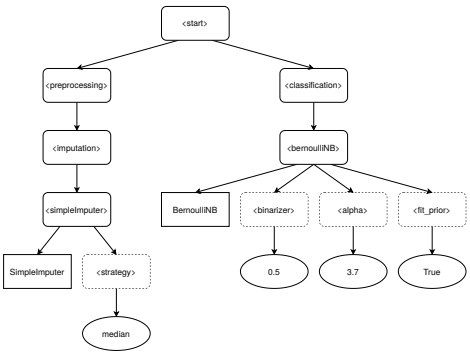
3.2.1 Neighborhood

Let S be a machine learning pipeline. The neighborhood $N(S)$ of S is the set of all pipelines that can be generated from S by randomly applying a mutation operator on one of its nodes. The mutation operator selects a node and replaces the subtree rooted at it with another subtree generated by the grammar [1]. Figure 3.3 repeats the pipeline from Figure 3.1 (Figure 3.3a) and shows two of its neighbors (Figures 3.3b and 3.3c). Gray-shaded nodes represent the subtree which was replaced by the mutation operator.

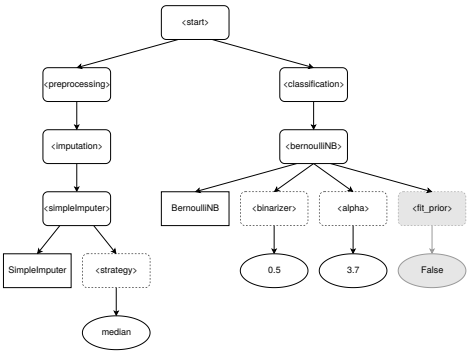
Figure 3.3b shows a pipeline with a single mutated hyperparameter, whereas the pipeline in Figure 3.3c had the whole preprocessing subtree replaced. We note that the impact of changing an algorithm, or even a complete subtree, may be much bigger than

Figure 3.3: Two neighbors of the pipeline from Figure 3.1. Gray subtrees indicate mutation points.

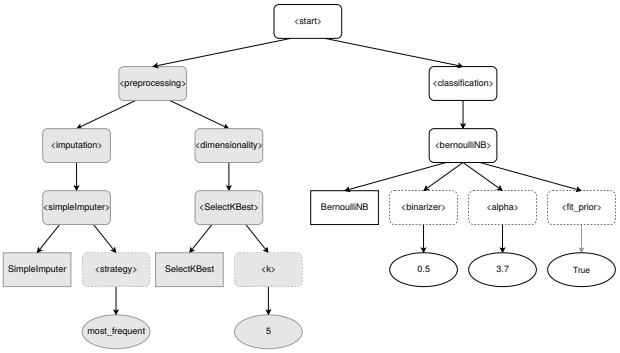
(a) Pipeline from Figure 3.1.



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



(c) Whole preprocessing subtree mutated.



Source: author.

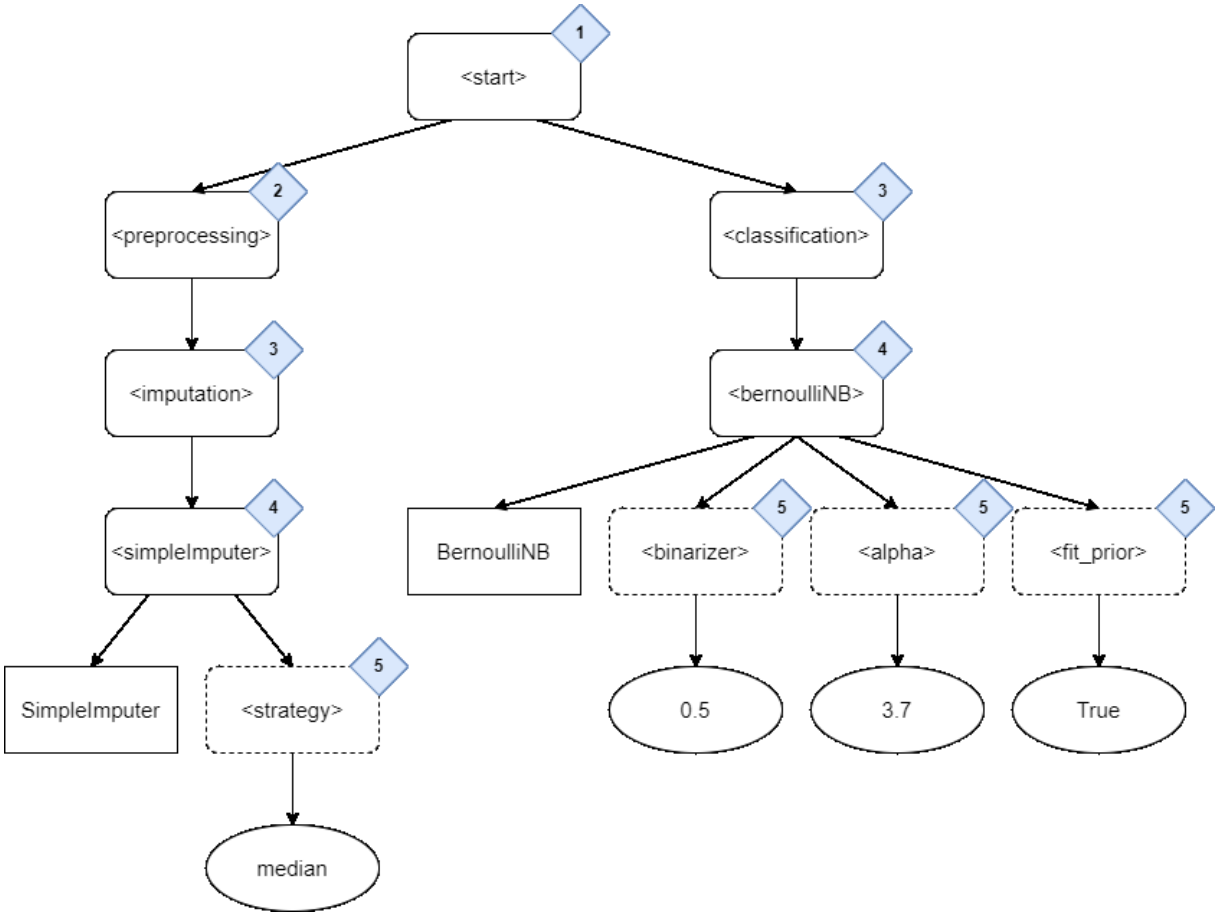
the impact of changing the value of a single hyperparameter, so we define the probability $p(x)$ of choosing node x from pipeline S as the mutation point to be proportional to the distance from the root of S. The exceptions to this rule are the leaves of the tree, which represent algorithm names or hyperparameter values. Such nodes are only changed when their parent node is selected as the mutation point. In order to do so, we give a weight $w(x)$ to each node that is directly proportional to the probability of choosing it, as defined in Equation 3.1.

$$w(x) = \begin{cases} \text{tree level} & \text{if non-terminal symbol} \\ 0 & \text{if terminal symbol} \end{cases} \tag{3.1}$$

The diamond shapes in Figure 3.4 show the weights of each symbol in the pipeline from Figure 3.1. The tree root (`<start>` symbol) has a weight of one. The preprocessing symbol has a weight of two, whereas the preprocessing subgroups and the classification symbol, a weight of three. The non-terminals representing algorithm names have a weight of four and those representing hyperparameter names have a weight of five. The probability $p(x)$ is then given by Equation 3.2. Considering this weighting scheme, the

neighborhoods generated by the mutation operator focus more on exploitation, rather than on exploration.

Figure 3.4: Pipeline with mutation weights.



Source: author.

$$p(x) = \frac{w(x)}{\sum_{x \in S} w(x)} \tag{3.2}$$

This definition of neighborhood is an extension of the one proposed by Garciarena et al. [25]. In their approach, two pipelines are neighbors if they differ in a single algorithm or hyperparameter. On the other hand, our approach is more closely related to the typical mutation operation used in grammar-based genetic programming [49]. In our representation, there is a probability, albeit small, of selecting the root of the tree and, thus, changing the whole pipeline. It is also possible to change all preprocessing steps of a pipeline at once, for example, something the original approach did not allow.

Note that this definition of neighborhood is a probability distribution over the complete search space. All experiments in Chapter 4 that require the calculation of the neighborhood of a pipeline use a random sample of this distribution.

3.2.2 Distance Between Pipelines

When trying to estimate the distance between machine learning pipelines, although not strictly mandatory, it is important to take into consideration the impact on the final result of the pipeline (the accuracy of a classification pipeline, for example). This takes us back to the previous discussion that replacing an algorithm usually has a bigger impact than simply modifying the value of a hyperparameter (note that this is not universally true, depending on the dataset, algorithms and hyperparameters, for example). Furthermore, there are several different types of machine learning algorithms. In this work, we grouped algorithms according to the classification of the Python library scikit-learn [12]. We hypothesize that replacing an algorithm with another from a different class will have a bigger impact on performance than replacing it with a more similar algorithm from the same class[1].

Thus, in the context of ML pipelines, the distance $dist(S_a, S_b)$ between trees S_a and S_b, which represent pipelines, must take into consideration the semantics of the difference between them. For this reason, determining the distance between pipelines is not straightforward and depends on expert knowledge. In order to define these distances, we classified the grammar symbols into 17 disjoint sets $A_0, A_1, \ldots A_{16}$. Table 3.2 describes the partitions containing special symbols, non-terminals and hyperparameters. The partitions of the preprocessing algorithms are shown in Table 3.3, whereas the classifiers are described in Table 3.4. All algorithms are based on the scikit-learn API[2] [12; 60].

Table 3.2: Symbol partitioning for special symbols, non-terminals and hyperparameters.

Partition	Description
A_0	*NULL* symbol
A_1	*<start>* symbol
A_2	*<preprocessing>* symbol
A_3	*<classification>* symbol
A_{14}	Discrete hyperparameters
A_{15}	Continuous hyperparameters
A_{16}	Categorical hyperparameters

The set A_0 is reserved for 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 between two symbols $x \in A_i$ and $y \in A_j$, which are nodes in the tree representation of the pipelines, is given by $d(x, y) = C$, where C is a constant that

[1]This observation is very sensitive to the way algorithms are grouped, which is left as future work.

[2]The documentation for the version of the API used in this work can be found at https://scikit-learn.org/0.22/modules/classes.html.

Table 3.3: Symbol partitioning for preprocessing algorithms.

Partition	Description	Algorithms
A_4	Imputation	SimpleImputer
A_5	Data range manipulation	Normalizer MinMaxScaler MaxAbsScaler RobustScaler StandardScaler QuatileTransformer Binarizer
A_6	Dimensionality manipulation and feature construction	VarianceThreshold SelectKBest UnivariateSelect SelectPercentile PCA IncrementalPCA KernelPCA FastICA GaussianRandomProjection SparseRandomProjection FeatureAgglomeration RBFSampler Nystroem TruncatedSVD LinearSVCPreprocessing ExtraTreesPreprocessing PolynomialFeatures BernoulliRBM RandomTreesEmbedding

depends on the class a symbol belongs to. If x and y have the same label, $d(x, y) = 0$. The proposed values of C are shown in Table 3.5.

In order to make the distance analyses possible, we impose the restriction that all pipelines must have at most three preprocessing algorithms (with the exception of pipelines designed for datasets with missing values, which always have an additional imputation algorithm) and exactly one classifier. Thus, we represent a tree S_i with root r_i as $S_i = r_i(c_1^{(i)}, c_2^{(i)}, \ldots, c_m^{(i)})$, 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 the root of a subtree. Let us consider S_a as the pipeline in Figure 3.1. r_a corresponds to the *<start>* node. It has two children, $c_1^{(a)}$ and $c_2^{(a)}$, which correspond to the nodes *<preprocessing>* and *<classification>*, respectively. We define a function $ch(x)$ that returns the first child of node x. In this example, the *<imputation>* group is denoted as $ch(c_1^{(a)})$, whereas *<bernoulliNB>* is given by $ch(c_2^{(a)})$.

The distance between two trees S_a and S_b will initially depend on whether they

Table 3.4: Symbol partitioning for classification algorithms.

Partition	**Description**	**Algorithms**
A_7	Naïve Bayes	GaussianNB BernoulliNB MultinomialNB ComplementNB
A_8	Linear models	SVC NuSVC LinearSVC LogisticRegression Perceptron PassiveAggressiveClassifier SGDClassifier RidgeClassifier RidgeClassifierCV
A_9	Neural networks	MLPClassifier
A_{10}	Nearest neighbors	KNeighborsClassifier RadiusNeighborsClassifier NearestCentroidClassifier
A_{11}	Discriminant analysis	LinearDiscriminantAnalysis QuadraticDiscriminantAnalysis
A_{12}	Trees	DecisionTreeClassifier ExtraTreeClassifier
A_{13}	Ensembles	RandomForestClassifier ExtraTreesClassifier AdaBoostClassifier GradientBoostingClassifier

Table 3.5: The proposed 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	0	0	0	0	0	0	0	0	0	0
A_7	0	0	0	0	0	0	0	1	2	2	2	2	2	2	0	0	0
A_8	0	0	0	0	0	0	0	2	1	2	2	2	2	2	0	0	0
A_9	0	0	0	0	0	0	0	2	2	1	2	2	2	2	0	0	0
A_{10}	0	0	0	0	0	0	0	2	2	2	1	2	2	2	0	0	0
A_{11}	0	0	0	0	0	0	0	2	2	2	2	1	2	2	0	0	0
A_{12}	0	0	0	0	0	0	0	2	2	2	2	2	1	2	0	0	0
A_{13}	0	0	0	0	0	0	0	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

include preprocessing steps or only a classification algorithm. Equation 3.3 shows four possible cases: neither S_a nor S_b 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 S_a (**C3**) or S_b (**C4**) have a preprocessing step, so we calculate the distance between the classification subtrees and add a constant k to the distance, where $k = d(\texttt{<preprocessing>}, NULL)$ is the distance between the *<preprocessing>* non-terminal and the *NULL* symbol, which is greater than the distance between any two preprocessing algorithms.

$$dist(S_a, S_b) = \begin{cases} dist_{clf}(ch(c_1^{(a)}), ch(c_1^{(b)})) & \textbf{C1} \\ dist_{pre}(c_1^{(a)}, c_1^{(b)}) + dist_{clf}(ch(c_2^{(a)}), ch(c_2^{(b)})) & \textbf{C2} \\ k + dist_{clf}(ch(c_1^{(a)}), ch(c_2^{(b)})) & \textbf{C3} \\ k + dist_{clf}(ch(c_2^{(a)}), ch(c_1^{(b)})) & \textbf{C4} \end{cases} \tag{3.3}$$

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 $children(x)$ is a function that returns all the children of node x. Sets A and B are initialized with the preprocessing groups of trees S_a and S_b, respectively (line 3). The first component of the distance is calculated as the distance from all groups that are present in only 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 S_a and S_b are different, we add their distance to the total distance (line 11). If they are the same, we add the distance between hyperparameter partitions, according to Table 3.5, to the total distance (lines 13 and 14), which is then returned in line 17.

As an example, consider that Algorithm 1 receives the pipelines from Figure 3.1 and Figure 3.3c as S_a and S_b, 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 S_a and S_b 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.

Equation 3.4 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_a and r_b, 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

Algorithm 1 Distance between preprocessing subtrees

procedure $\text{DIST}_{pre}(S_a, S_b)$ ▷ The root of S_a and S_b is the *<preprocessing>* symbol
  $distance \leftarrow 0$
  $A \leftarrow \{n \mid n \in children(S_a)\}$; $B \leftarrow \{n \mid n \in children(S_b)\}$
  $diff \leftarrow (A - B) \cup (B - A)$
  $distance \leftarrow distance + \sum_{i=1}^{|diff|} d(diff_i, NULL)$
  $intersect \leftarrow A \cap B$
  for all $label \in intersect$ **do**
    $algA \leftarrow ch(get\_node(label, A))$
    $algB \leftarrow ch(get\_node(label, B))$
    if $algA \neq algB$ **then** ▷ Different algorithms
      $distance \leftarrow distance + d(algA, algB)$
    else ▷ Same algorithm; check hyperparameters
      $hpA \leftarrow children(algA)$; $hpB \leftarrow children(algB)$
      $distance \leftarrow distance + \sum_{i=2}^{|hpA|} d(ch(hpA_i), ch(hpB_i))$
    end if
  end for
  return *distance*
end procedure

distance between the values of the hyperparameters of each algorithm. If the algorithms are not the same, the distance between the trees is the distance between the partitions to which each algorithm belongs, given in Table 3.5.

$$dist_{clf}(S_a, S_b) = \begin{cases} \sum_{j=2}^{m} d(ch(c_j^{(a)}), ch(c_j^{(b)})) & \text{if } r_a = r_b, \\ d(r_a, r_b) & \text{otherwise.} \end{cases} \tag{3.4}$$

3.3 Fitness Function

The last component of a fitness landscape is a fitness function $f : X \to \mathbb{R}$ that assigns a real number to each element of the set of configurations X, described in Section 3.1. In this work, we deal with multiclass classification problems with class imbalance. Therefore, we use the F1 score [88] to evaluate the quality of the solution given by each machine learning pipeline. Equation 3.5 defines F1 score for binary classification problems, where TP, FP and FN are the numbers of true positives, false positives, and false negatives given by the pipeline, respectively. When working with classification problems with class imbalance, F1 score is a useful way of measuring the model's accuracy, because it not only considers the errors made, but also the types of errors, revealing models that always predict the majority class, for example.

$$F1\ score = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{3.5}$$

As we deal with multiclass classification, we use a one-vs-all approach, in which we consider the results of a binary classifier for each class of the dataset. We then take the weighted average of the F1 score for each classifier as the final pipeline's fitness.

3.4 Adaptations of FLA Metrics

This section describes the adaptations that had to be made in order to apply Fitness Landscape Analysis to the context of AutoML.

3.4.1 Random Walks

Many FLA metrics depend on random walks over the landscape, either because the original formulation requires such a walk, or because computational complexity would prevent the exploration of the complete search space. Several random walk algorithms have been proposed with the goal of sampling discrete or continuous landscapes, such as adaptive walks [36], neutral walks [68], progressive walks [47] and distributed walks [40]. In this work, we used a simple random walk, as described below.

Given a starting point (i.e., a machine learning pipeline), we generate a random neighborhood of size n by applying the mutation operator, described in Section 3.2.1, on the starting point n times, randomly select a neighbor and use it as the next starting point. We repeat this process until a desired walk length is achieved or a number of invalid pipelines that is five times the walk length is generated. Pipelines are invalid if the model throws an exception during training (this might happen when a forbidden combination of hyperparameter values is not captured by the grammar, or the eigenvalue computation does not converge when running Principal Component Analysis, for example) or training exceeds the time budget of 300 seconds per pipeline.

In order to perform random walks, we could apply a mutation operator a single time to generate a random neighbor. However, we opted to generate a random neighborhood and then choose a random neighbor as the next step to be able to use the same experiment to calculate both correlation length and neutrality.

3.4.2 Neutrality for Continuous Fitness

The definition of neutrality given in Section 2.2.1 expects a discrete fitness in order to check for equality. In the case of classification pipelines, we use the F1 score, which is continuous, as the fitness function. Thus, we adapted the definition of neutrality to account for pipelines that are neutral within a specified tolerance δ for the fitness value. Equation 2.4, which defines the neutral neighborhood $\mathcal{N}(s)$ of a pipeline s considering a discrete fitness function, can be rewritten as Equation 3.6.

$$\mathcal{N}^{\approx}(s) = \{s' \in N(s) \mid |f(s') - f(s)| < \delta\}, \tag{3.6}$$

where $\mathcal{N}^{\approx}(s)$, $N(s)$ and $f(s)$ are the quasi-neutral neighborhood, a sample of the complete neighborhood, and the fitness of s, respectively, and $\delta \geq 0$ is a small constant. The neutrality ratio of s can then be rewritten as $|\mathcal{N}^{\approx}(s)|/|N(s)|$.

3.4.3 Correlation Length for Non-isotropic Landscapes

The autocorrelation function used to calculate correlation length assumes that the landscape is statistically isotropic, which means that the statistics of the fitnesses of a random walk are the same, regardless of the starting point. To the best of our knowledge, this might not be the case for AutoML landscapes. Thus, we performed several random walks with different starting points in order to obtain more significant values for the correlation length metric [32].

3.4.4 FDC for Very Large Search Spaces

In its original formulation, given in Section 2.2.3, FDC requires knowledge of the global optimum (or optima). Malan and Engelbrecht [46] proposed an adaptation of FDC, dubbed the fitness distance correlation searchability, or FDC_s, for continuous problems, for which the global optimum is not known.

Given a random sample of n points from the search space, the fitness values $F = \{f_1, \ldots, f_n\}$ are evaluated and their mean $\overline{f}$ is calculated. The distance from the best

configuration in the sample to every point, as described in Section 3.2.2, is denoted by $D^* = \{d_1^*, \ldots, d_n^*\}$, with mean $\overline{d^*}$. FDC_s is given by Equation 3.7. From here on, we denote FDC_s simply by FDC.

$$FDC_s = \frac{\sum_{i=1}^{n}(f_i - \overline{f})(d_i^* - \overline{d^*})}{\sqrt{\sum_{i=1}^{n}(f_i - \overline{f})^2}\sqrt{\sum_{i=1}^{n}(d_i^* - \overline{d^*})^2}} \tag{3.7}$$

3.5 AutoML Optimizers

In order to evaluate how AutoML optimizers deal with search spaces with different characteristics, we implemented two algorithms from the literature: TPE [4] and SMAC [29]. Both algorithms follow the Sequential Model-Based Optimization (SMBO) approach. SMBO is a framework based on Bayesian optimization for solving algorithm configuration problems, which can also be applied to the problem of generating complete machine learning pipelines [23; 78]. The framework consists of iterating between fitting regression models that predict performance and using the results to choose promising configurations from unseen regions of the parameter space [28; 29].

SMAC (Sequential Model-based Algorithm Configuration) is an instantiation of the SMBO framework which supports numerical, categorical and conditional hyperparameters and can be used for sets of problem instances [29]. SMAC uses as its surrogate a model based on the idea of random forests, which have good performance for categorical data. The random forest is constructed as a set of regression trees using a given number of observations sampled randomly from the training set with repetition. SMAC then obtains a predictive mean and variance of a new configuration as the empirical mean and variance of its trees' predictions for such configuration. Promising hyperparameter configurations are selected by optimizing an acquisition function, known as the Expected Improvement (EI) [33], over the best configuration found so far – also called the incumbent. This is done by performing a multi-start local search, chosen from the previous iterations, and considering all configurations with locally optimal EI, which can be evaluated based on the mean and variance of the predictions.

Another crucial step of SMAC is deciding how many evaluations should be performed for each configuration and when a configuration can be chosen as the incumbent. This component is known as the intensification mechanism. SMAC does this by taking a list of promising configurations, chosen based on EI, and comparing them to the current incumbent until a time budget is reached. In order for a configuration to become the new incumbent, it must outperform the current incumbent in every sample of the data,

making SMAC robust for noisy function evaluations.

TPE, in turn, uses tree-structured Parzen estimators (TPE) as the surrogate model and also relies on EI as its acquisition function [4]. TPE uses Bayes' rule to define the probabilistic model. Using the notation defined in Section 2.1 and considering maximization problems, TPE defines two non-parametric densities for the prior configuration, as shown in Equation 3.8.

$$p(\lambda \mid y) = \begin{cases} g(\lambda) & \text{if } y > y^* \\ l(\lambda) & \text{if } y \leq y^* \end{cases}, \tag{3.8}$$

where $y = \mathcal{G}(A, \mathcal{D}^{(i)}_{\text{train}}, \mathcal{D}^{(i)}_{\text{valid}})$, $g(\lambda)$ is the density generated by using configurations that resulted in a gain greater than a threshold y^* and $l(\lambda)$ is the density generated by the remaining observations. y^* is chosen as some quantile γ of the observed values of y, such that $p(y < y^*) = \gamma$.

The work done by Bergstra et al. [4] shows a closed form of the EI equation that is proportional to $g(\lambda)/l(\lambda)$[3]. Therefore, in order to maximize EI, TPE draws candidates λ that are more likely under $g(\lambda)$ than under $l(\lambda)$ and evaluates them in terms of $g(\lambda)/l(\lambda)$, returning the candidate with the greatest EI at each iteration of the algorithm.

Following the SMBO framework, at each iteration, the actual fitness function is evaluated for the chosen candidate and used to update the surrogate model. When the time or iterations budget is exceeded, the best candidate is returned.

We executed the optimizers for the three smallest search spaces, namely `small`, `default` and `auto-sklearn`, and evaluated the best fitness found in each iteration of the algorithm, the number of distinct pipelines and the percentage of the search space explored. We also calculated FDC for the optimizers, using the explored pipelines as the random sample required by the metric.

[3]Note that the original formulation treats hyperparameter optimization as a minimization problem, so the authors describe EI as inversely proportional to this ratio.

Chapter 4

Experimental Evaluation and Results

In this chapter, we present the experimental evaluation and discuss the results. We initially evaluated the fitness of random samples of the search spaces in order to obtain preliminary insights into the difficulty of the classification tasks (i.e., the existence of good solutions), which does not necessarily reflect the difficulty of finding the best machine learning pipeline to solve such tasks. We also calculated the global optimum for the two smallest search spaces, namely `small` and `medium`. We then calculated the FLA metrics Neutrality, Correlation Length and FDC for all search spaces and datasets. Lastly, we implemented AutoML optimizers based on TPE and SMAC, evaluated their performances on the proposed search spaces and calculated FDC for the explored regions. Pipeline evaluations were performed using a 5-fold cross-validation procedure, except for the SMAC and TPE executions, which performed a single training step with the entirety of the training data. Fitness values used to calculate FLA metrics and reported in the form of tables were obtained for a separate test set.

4.1 Datasets

In order to evaluate the characteristics of the search spaces described in Section 3.1, we used nine datasets, obtained from OpenML [84]. Table 4.1 summarizes their main characteristics, including a link to the source (note that some of the original names have been modified), the number of instances (# inst.), features (# feat.) and classes (# classes) and information regarding the presence of missing values.

Most of the datasets are part of the OpenML-CC18 benchmarking suite [7], which fulfills several useful requirements for comprehensive and practical benchmarking. The exceptions are *statlog-segment*, which is a part of OpenML100 [6], a predecessor of OpenML-CC18, and *wine-quality-red*, which is not a part of any OpenML benchmarks. In order to simplify the evaluation of the machine learning pipelines, we removed all categorical features, otherwise the pipelines would need to include encoding steps for such features.

Table 4.1: Datasets used in the experiments.

Dataset	# inst.	# feat.	# classes	Missing	Observation
breast-w	699	9	2	Yes	
diabetes	768	8	2	No	
mice-protein	1,080	77	8	Yes	Removed categorical features
ml-prove	6,118	51	6	No	
statlog-segment	2,310	19	7	No	
texture	5,500	40	10	No	500 instances missing
vehicle	846	18	4	No	
wilt	4,839	5	2	No	
wine-quality-red	1,599	11	6	No	

4.2 Fitness Landscapes

For each of the selected datasets, we built a fitness landscape for each search space defined in Chapter 3. For that, we need to evaluate all candidate solutions using the F1 score, defined as our fitness. However, it is usually unfeasible to completely enumerate the search spaces in order to obtain their global optima, but we were able to evaluate all pipelines in search spaces `small` and `medium`, which were built with the purpose of being toy examples to our analysis.

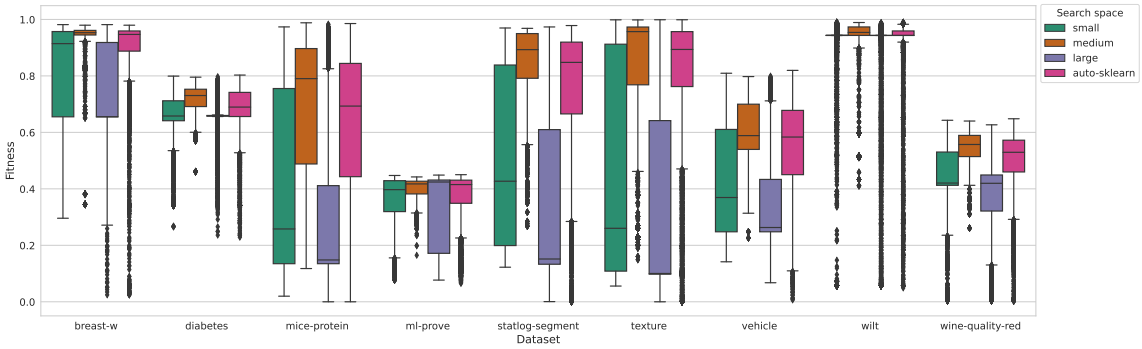
Table 4.2 shows the best fitness value for each combination of the two enumerable search spaces and dataset and the number of pipelines with that fitness value, with a precision of three decimal places. This gives us the number of global optima. We see cases in which a large number of pipelines with the same approximated optimal fitness (e.g., dataset mice-protein on search space `medium`), whereas other combinations have a single optimum (e.g., datasets breast-w, ml-prove and vehicle on search space `small`). Despite the differences in the number of global optima, the best fitness values are very similar for both search spaces, usually differing in less than 1% (the exceptions are datasets vehicle and wine-quality-red, with differences of approximately 1.5% and 3%, respectively). It stands to reason that, fixing all other characteristics of the search space, it may be easier for an optimizer to find an optimal solution when there are several of them. Note that this does not mean that the search space is easier, only that it is easier for an optimizer to find a global optimum by chance.

After finding the optima for the two enumerable search spaces, in order to obtain an overall understanding regarding the difficulty of the classification task for each dataset, we generated the fitness of a random sample of 20,000 pipelines from each of the four search spaces. Figure 4.1 shows a boxplot of the results, while Table 4.3 summarizes the mean and standard deviation (within parentheses) of the fitness.

Table 4.2: Global optima of each combination of search space and dataset.

Search space	Dataset	Best fitness	# pipelines
small	breast-w	0.982	1
	diabetes	0.803	2
	mice-protein	0.974	10
	ml-prove	0.446	1
	statlog-segment	0.968	2
	texture	0.999	10
	vehicle	0.811	1
	wilt	0.990	6
	wine-quality-red	0.625	8
medium	breast-w	0.978	54
	diabetes	0.801	3
	mice-protein	0.983	990
	ml-prove	0.448	33
	statlog-segment	0.970	40
	texture	0.998	6
	vehicle	0.799	3
	wilt	0.989	3
	wine-quality-red	0.645	16

Figure 4.1: Fitness evaluation of a random sample of the search space.



Source: author.

Table 4.3: Best fitness of a random sample of the search space. Values within parentheses show the difference between the sample optimum and the global optimum.

Dataset	small	medium	large	auto-sklearn
breast-w	**0.982** (0.0%)	0.978 (0.0%)	**0.982**	0.980
diabetes	0.799 (0.5%)	0.796 (0.6%)	0.797	**0.803**
mice-protein	0.974 (0.0%)	0.983 (0.0%)	0.983	**0.985**
ml-prove	0.445 (0.2%)	0.442 (1.3%)	0.449	**0.450**
statlog-segment	0.965 (0.3%)	0.968 (0.2%)	0.973	**0.978**
texture	**0.999** (0.0%)	0.998 (0.0%)	**0.999**	**0.999**
vehicle	0.809 (0.2%)	0.798 (0.1%)	0.799	**0.820**
wilt	**0.990** (0.0%)	0.989 (0.0%)	0.989	**0.990**
wine-quality-red	0.623 (0.3%)	0.640 (0.8%)	0.627	**0.648**

For search spaces `small` and `medium`, the random sample, albeit small, is able to capture the global optimum or find solutions with fitness very close to the optimum. For all datasets, except for breast-w, the sample of search space `auto-sklearn` contains the best solutions. When we only consider search spaces `small`, `medium` and `large`, which were specifically proposed for this work, `small` contains the best solution for five out of nine datasets, `medium` for two datasets and `large` for six[1]. Although it may appear that search spaces with more algorithms (i.e., `small` and `large`) are better than search spaces with fewer algorithms (i.e., `medium`), we note that the fitness values of the best solution found in all search spaces are very close. An implication of this observation is that, in general, smaller search spaces might be sufficient and, depending on how the chosen optimization algorithm explores the space, easier to search.

4.3 FLA Metrics

After obtaining some hints on the difficulties of the landscapes generated by the four search spaces, this section shows the results of Neutrality, FDC and Correlation Length, which address Research Question 1. These metrics were calculated using a sample of the search space, obtained uniformly at random or using the random walk described in Sections 3.4.1.

4.3.1 Neutrality

For the analysis of the neutrality of the search space, for each combination of search space and dataset, we performed ten random walks of length up to 15,000 pipelines, starting from random points in the search space. For each solution in the walk, we evaluated the fitness function of up to 50 randomly generated neighbors and analyzed its neutrality ratio. One of the neighbors was randomly selected to be the next point in the walk. Here we report the mean neutrality ratio of the complete walk.

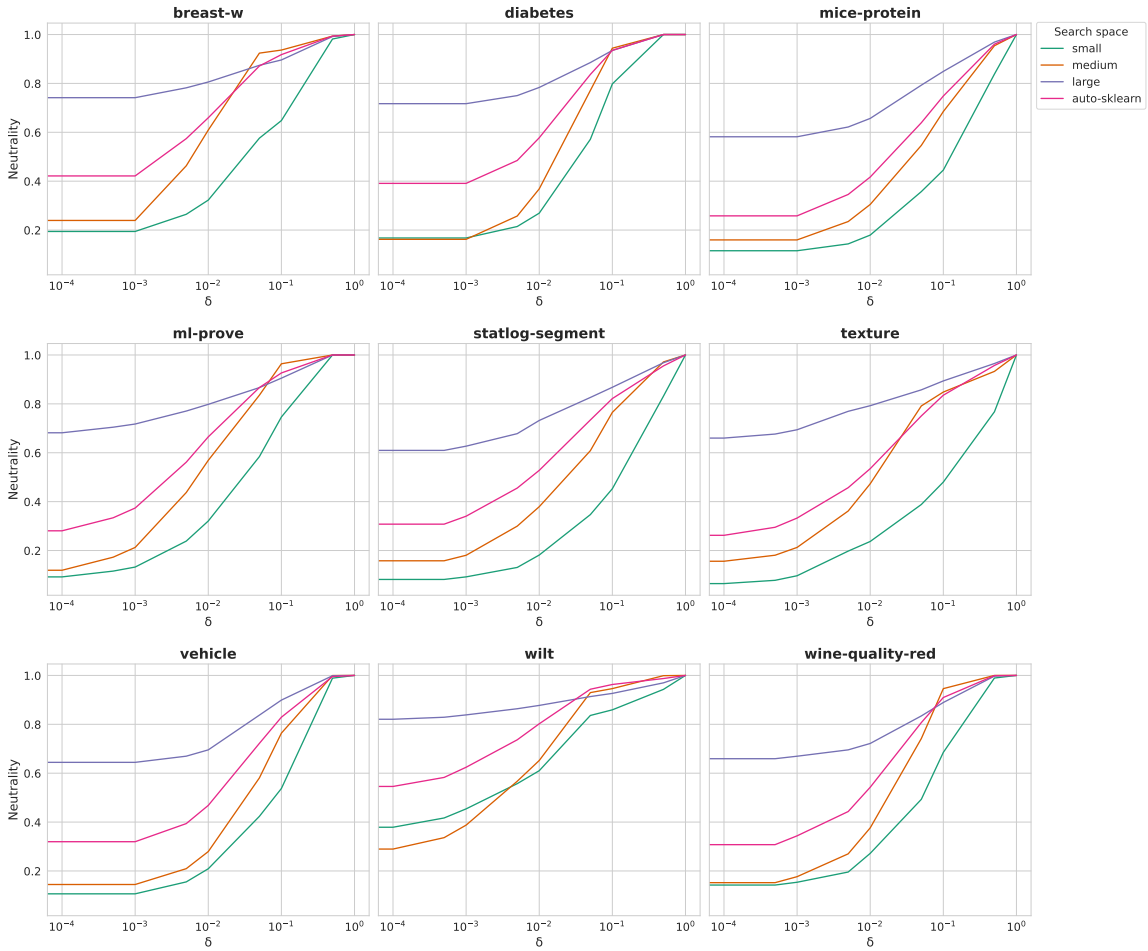
We defined both the length of the random walks and the size of the neighborhood in terms of valid pipelines. When the experiment exceeded five times the walk length without finding the desired number of valid pipelines, the walk was interrupted at the

[1]More than one search space may contain pipelines with the best fitness for each dataset.

current length. A similar approach was used for the generation of the neighborhood, with a limit of ten times the size of the neighborhood for each point in the random walk.

Recall from Section 3.4 that, for continuous fitness functions, we consider neutrality within a specified tolerance δ. We executed experiments varying the values of δ from 0 to 1, as shown in Figure 4.2. As expected, neutrality increases for higher tolerance thresholds. Furthermore, the behavior of neutrality as a function of δ varies between search spaces and datasets. Therefore, we set the tolerance independently for each combination of dataset and search space as the standard deviation of the mean fitness of the random sample of size 20,000. However, it might be more appropriate to define the tolerance according to the business needs of the application, in terms of fitness requirements, which is beyond the scope of the present work.

Figure 4.2: Effect of the tolerance threshold δ on the neutrality ratio, considering the longest random walks and largest neighborhoods.

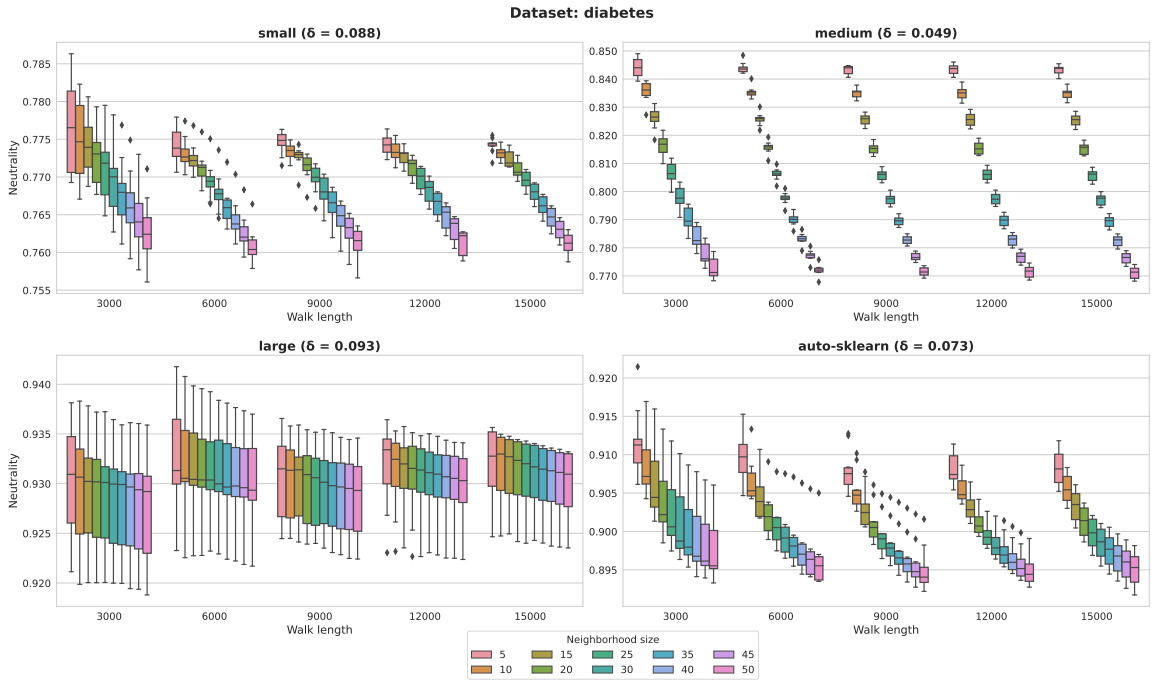


Source: author.

Figure 4.3 illustrates the neutrality results for the diabetes dataset. The results for the other datasets, which are mostly similar to those for the diabetes dataset, can be found in Appendix B.1, Figures B.1 to B.8. Search space `small` showed the lowest neutrality ratios for all datasets, except for wine-quality-red when evaluating larger neighborhood

sizes. This search space keeps all hyperparameters fixed at their default values, so the results suggest that changing hyperparameter values indeed has a lower impact on the fitness of the solution than changing algorithms. Search space `large`, which has the largest number of algorithms and hyperparameter values, showed the highest neutrality ratios for six datasets. The exceptions are datasets breast-w,ml-prove and wilt, for both the smallest and largest neighborhood sizes. We were unable to identify characteristics of such datasets that could explain this behavior. Note that higher neutrality ratios may indicate either a more globally neutral landscape or landscapes with larger neutral regions.

Figure 4.3: Neutrality evaluation for the diabetes dataset (note the different scales).



Source: author.

Overall, these results seem to confirm the expectation that larger search spaces, especially those with broad hyperparameter domains, tend to be more neutral. The possible effects of neutrality on the performance of optimization methods will be explored in Section 4.4. It is important to note that we evaluated methods based on Bayesian optimization, which make a global exploration of the search space. The performance of this kind of method may be less affected by the presence of neutrality than gradient-based methods or local search.

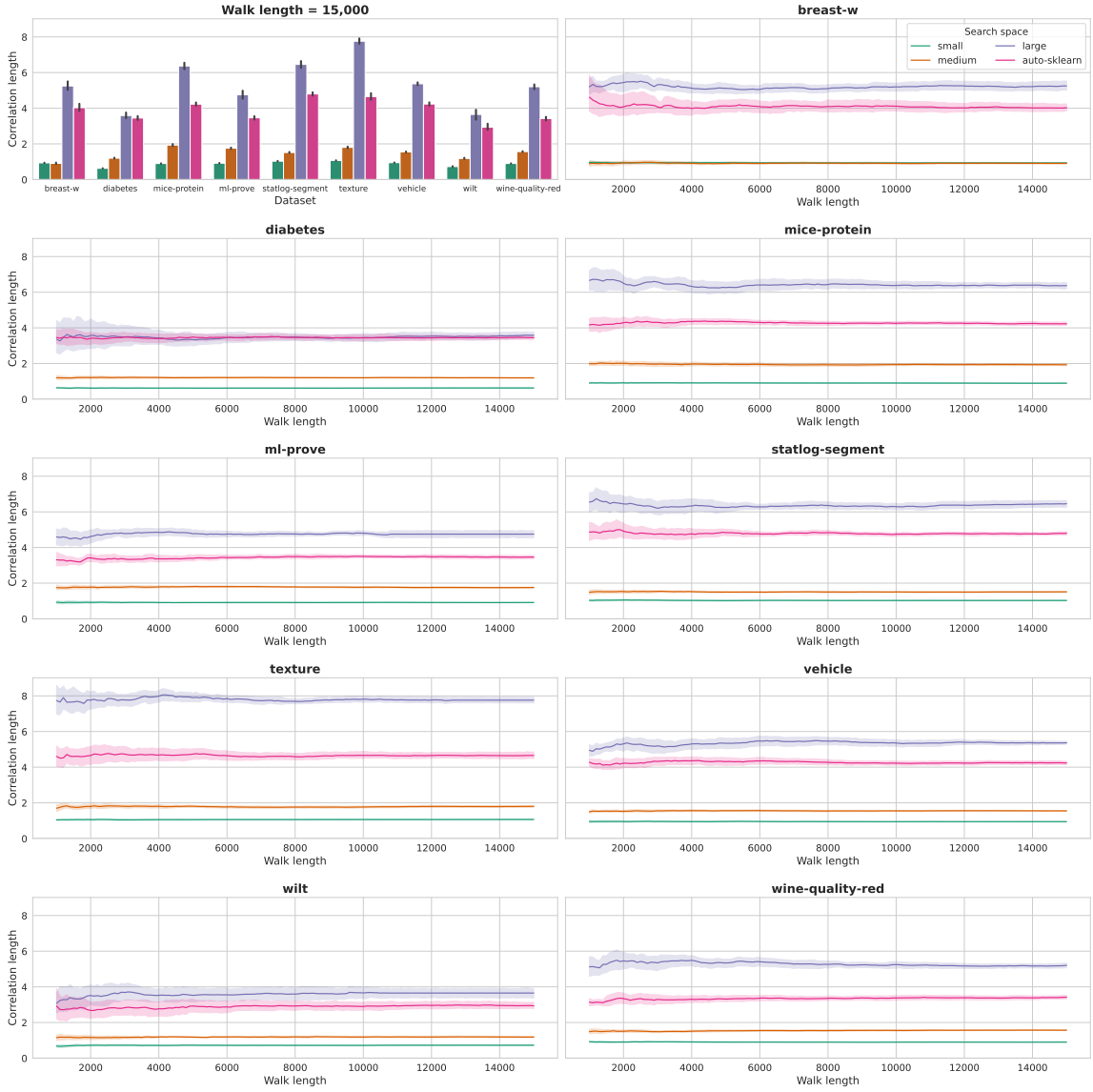
The grammars used to define the search spaces allow the existence of some conditional hyperparameters that are ignored by the algorithm based on the value of another hyperparameter. When this happens, pipelines that only differ in the value of a hyperparameter that is ignored are inherently the same, generating artificial neutrality. Although the grammars could be modified to prevent this from happening, we chose not to do so, given that it would require increasing the complexity of the grammar, which would affect

the way we represent the semantics of the pipelines, calculate the distance and define the neighborhood.

4.3.2 Correlation Length

Correlation Length is a way of measuring the ruggedness of a landscape and is based on random walks. For this reason, we calculated this measure using the neutrality experiments, described in Section 4.3.1, with the largest neighborhood size.

Figure 4.4: Correlation length results.



Source: author.

Figure 4.4 shows the correlation length for different walk lengths and a summa-

rization of the results for a walk of 15,000 pipelines. For walks of length up to 5,000, we see some instability in the correlation length for different executions of the random walk, but it becomes stable after this point. For seven datasets, correlation length is ordered, in ascending order, according to the size of the search space, meaning that `small` always has the smallest correlation length, meaning more rugged landscapes (see Table 2.1), followed by `medium`, `auto-sklearn` and `large`. For dataset breast-w, the ordering is maintained, although `medium` and `small` have roughly the same correlation length. For diabetes, this can be observed for search spaces `auto-sklearn` and `large`. Correlation length also depends on the characteristics of the dataset, although this effect is much less pronounced than what was observed for the search space.

4.3.3 Fitness Distance Correlation

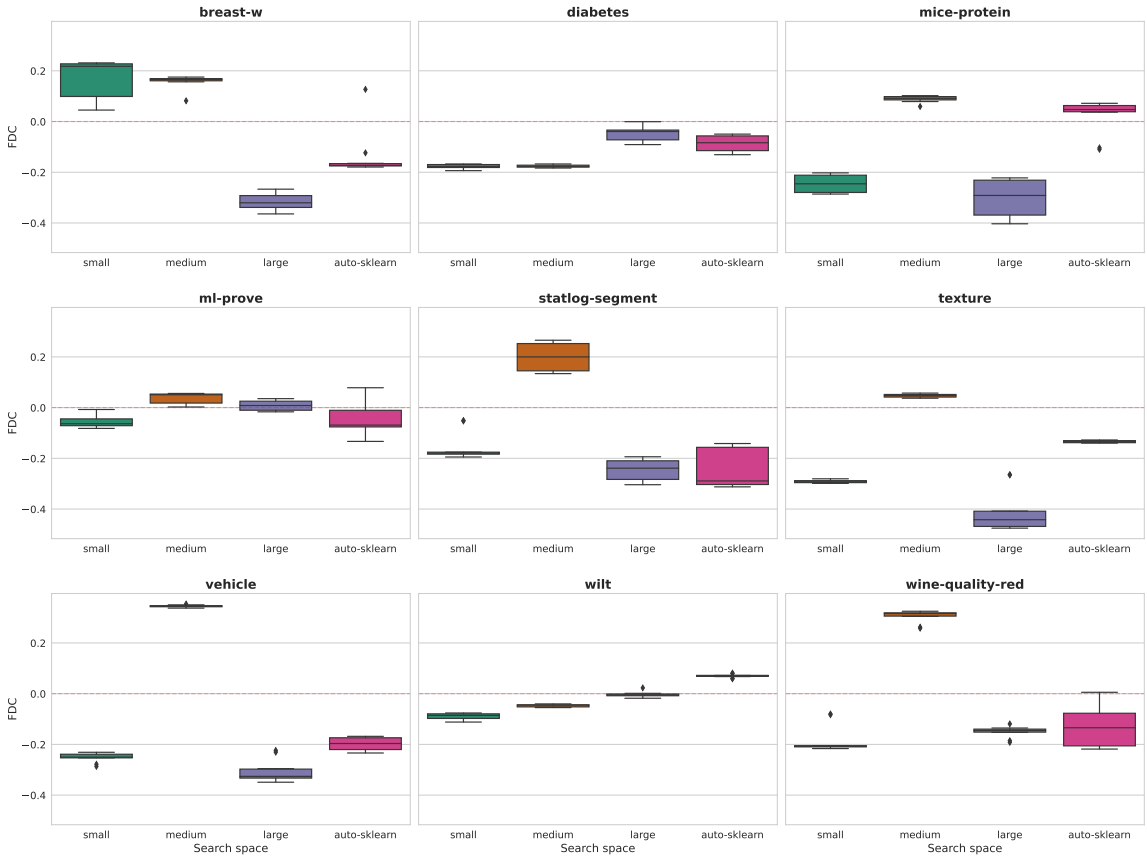
FDC analysis was carried out by evaluating ten independent random samples of 10,000 pipelines each. In our previous study [61], in which we evaluated samples of size 3,000, we demonstrated the sample size has little effect on FDC. Thus, we omitted this experiment here. Recall that, for maximization problems (see Equation 2.3), low FDC values are considered easy, values around zero are hard and high values indicate misleading landscapes. Some works proposed specific values that could be considered low or high, but we limit ourselves to the comparison of FDC values for different datasets and search spaces.

Figure 4.5 illustrates the results of the FDC analysis. As we can see, FDC is not consistent among datasets and search spaces. Any given dataset can produce positive or negative FDC values for different search spaces. The same can be observed when grouping the results by search space. Overall, 24 out of 36 combinations of dataset and search space had negative FDC values, which suggests easy landscapes, ten combinations are considered misleading by FDC and two are hard[2]. It is important to note that FDC tries to describe the searchability of the landscape, not the hardness of the classification problem. Thus, we can observe instances that are described as hard or misleading by FDC, but have high fitness values (Tables 4.2 and 4.3).

A major drawback of using FDC for continuous spaces is the dependence on the local optimum for a sample of the space, which may be arbitrarily worse than the global optimum. In order to assess the impact of this adaptation, we executed smaller FDC experiments, using ten independent random samples of size 5,000, using either the global or local optima for search spaces `small` and `medium`, whose global optima are known. We

[2]We considered a landscape as hard if the interquartile range of the boxplot included the value zero.

Figure 4.5: FDC results.



Source: author.

call these results FDC_g. Figure 4.6 shows considerable differences between the local FDC and FDC_g for dataset breast-w and search space `small` and for dataset statlog-segment and search space `medium`. In the former case, FDC indicates a misleading landscape, whereas FDC_g indicates that the landscape is easy. The latter is less extreme. The landscape was identified as misleading by FDC and hard by FDC_g. Besides these changes in interpretation, FDC_g also showed less variation than FDC. Despite these differences, the global optimum is rarely known, and the computational cost of evaluating machine learning pipelines makes it unfeasible to evaluate larger samples.

FDC results indicate that the searchability of AutoML problems greatly depends on the dataset and the search space, making it hard to generalize assumptions regarding problem difficulty. Thus, we believe that FDC should always be analyzed together with other landscape characteristics and only be interpreted as a weak indication of problem difficulty.

Figure 4.6: FDC results using global and local optima for completely explored search spaces.

Source: author.

4.4 AutoML Optimizers

In order to obtain some insights on how the characteristics of the fitness landscape impact the performance of search methods (Research Question 2), we executed ten independent runs of TPE and SMAC, with a time budget of five hours. Tables 4.4 and 4.5 describe the hyperparameters of the TPE and SMAC implementations, respectively. Note that the original implementations [4; 29] of both algorithms used a hyperparameter to control the number of iterations or executions, which we replaced with a time budget. We also omitted the extra evaluations of Expected Improvement (EI) for a random sample from SMAC, due to time constraints.

The number of pipelines evaluated within the time budget shows great variation for each execution of the algorithms. This can occur for a number of factors, such as the time required to train the pipelines and concurrence with other processes running on the same computer, for example.

For the TPE and SMAC executions, pipelines were evaluated using the same framework as the complete exploration of the search spaces and the computation of FLA metrics.

However, training was done using the complete training data, rather than using a 5-fold cross-validation procedure, due to time constraints. For this reason, fitness values are not directly comparable to those obtained for the global optima evaluation of enumerable search spaces. Despite this limitation, we were still able to evaluate how the optimizers respond to the characteristics of the landscapes, as identified by the metrics described in Section 4.3.

Table 4.4: Hyperparameters of the TPE implementation.

Hyperparameter	**Value**
Size of the random sample obtained before starting the algorithm	20
Number of points chosen from the EI distributions to be used in the evaluation of the objective function at each iteration	10^8
Fraction of best point from the previous iteration to be used to construct the new density functions	0.25

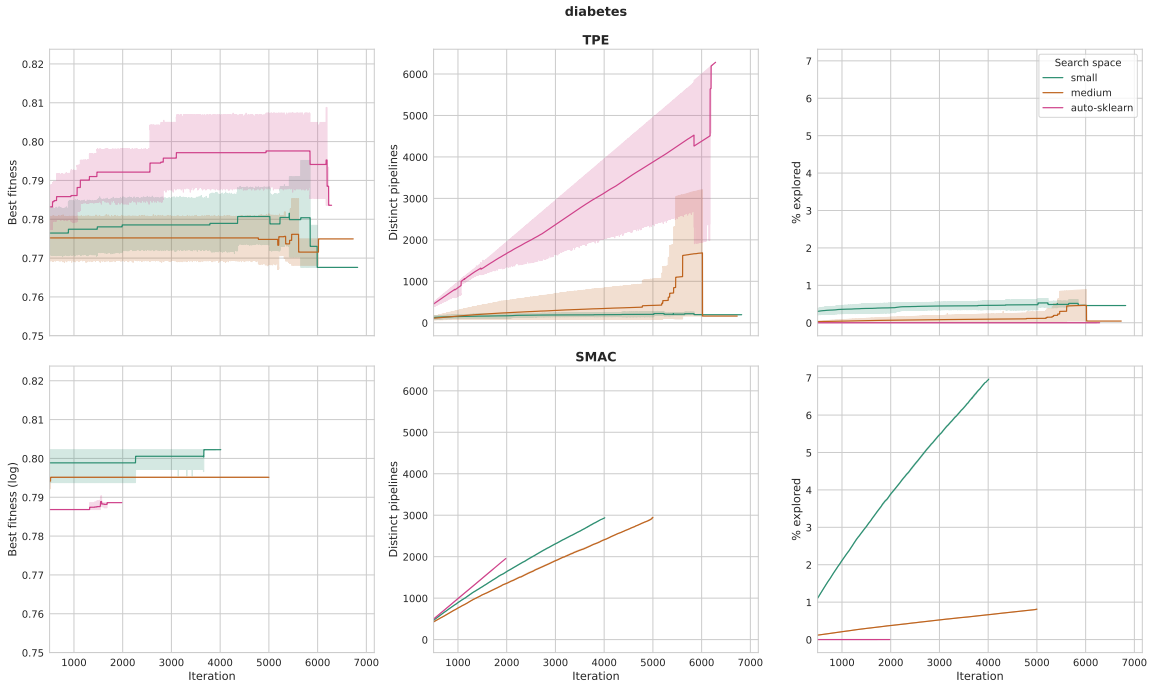
Table 4.5: Hyperparameters of the SMAC implementation.

Hyperparameter		**Value**
SMAC HPs	Percentage of promising configurations to use in the intensify step	0.2
	Size of the random sample used to evaluate EI during the local search step	10
Random Forest HPs	Number of trees in the forest	10
	Use bootstrap samples	*True*
	Maximum depth of the trees	2^{20}
	Minimum number of samples required to split an internal node	3
	Minimum number of samples required to be at a leaf node	3
	Number of features to consider when looking for the best split	5/6 of the features

Figure 4.7 summarizes the results for dataset diabetes, which is representative of other datasets[3]. The plots on the left-hand side show the best fitness found by each optimizer at each iteration of the algorithm. As we can see, SMAC executes fewer iterations given the same time budget as TPE, but both algorithms seem to converge after very few iterations. The smaller number of iterations performed by SMAC can be due to the overhead of evaluating random forests. For some datasets, TPE tends to find slightly better pipelines for search space `auto-sklearn`, which has the highest neutrality ratio and correlation length among the three evaluated search spaces. The plots in the middle

[3]The solid lines represent the mean of 10 independent runs. Given the variation in the number of evaluated pipelines in each run, the mean can decrease in later iterations.

Figure 4.7: Exploration of the search spaces by TPE and SMAC for dataset diabetes.



Source: author.

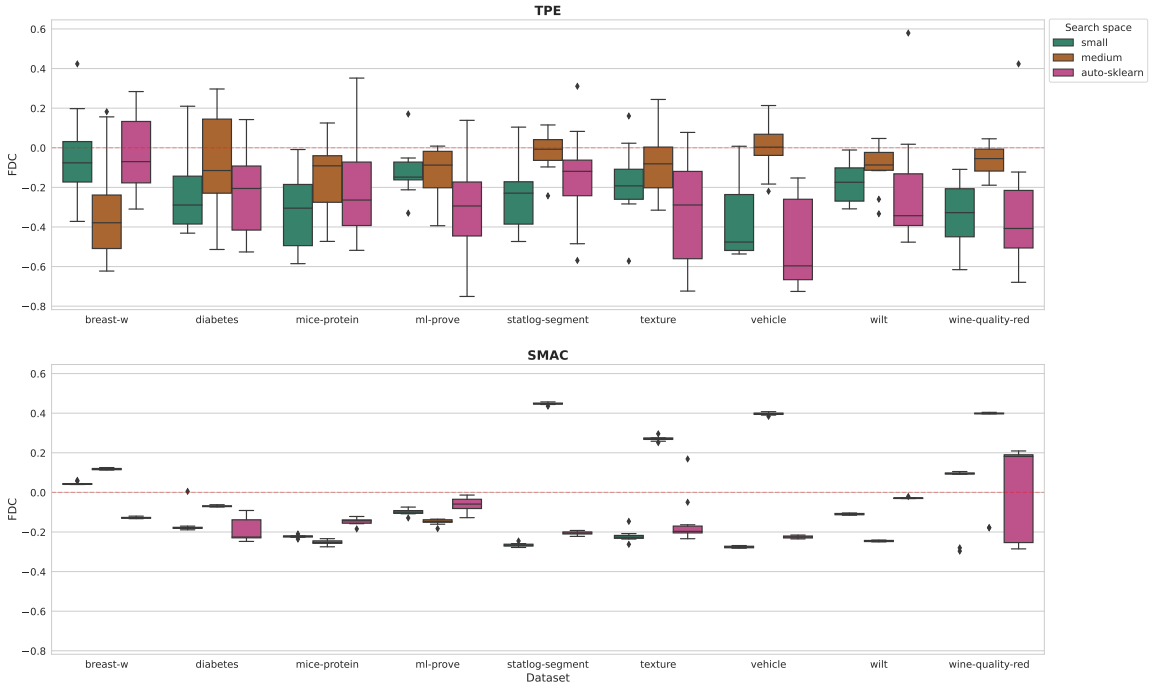
of the figure show how many distinct pipelines were evaluated at each iteration, whereas the right-hand side corresponds to the percentage of the search space that has been explored so far. TPE gets stuck in a very small number of pipelines for search spaces `small` and `medium`, but is able to better explore search space `auto-sklearn`. This suggests that TPE can benefit from the presence of neutrality, but struggles to explore more rugged landscapes. On the other hand, SMAC is able to efficiently explore all search spaces. In fact, for search space `auto-sklearn`, there is very little repetition of pipelines, followed by `small` and `medium`. This suggests that SMAC's performance is also hindered by an increase in ruggedness, but to a much lesser extent than TPE. Furthermore, `small` is more rugged than `medium`, but is better explored by SMAC, so there are probably other characteristics of the landscape affecting the exploratory ability of the algorithm. The results for the other datasets can be found in Appendix B.2, Figures B.9 to B.16.

4.4.1 FDC of the Explored Regions

We calculated FDC for the pipelines explored by TPE and SMAC, using the best solution found as the optimum. Figure 4.8 shows the results. As presented in Section 4.3.3, FDC values greatly differ between search spaces and datasets, so we were unable to find

meaningful patterns that could explain optimizer performance. However, when comparing TPE and SMAC, we note that, although both are able to find pipelines with good fitness, they explore the space in very different ways. For instance, there is a much larger variation in the results of the independent runs of TPE than those of SMAC. Furthermore, there are several instances in which the interpretation of problem difficulty given by FDC is different for each optimizer. For example, TPE's exploration of search space `medium` with dataset breast-w revealed an easy landscape (negative FDC), whereas SMAC identified it as slightly hard or misleading (small positive value). Similarly, the landscape generated by search space `medium` and dataset statlog-segment was identified as easy by SMAC and misleading by TPE. Moreover, FDC results vary a lot for TPE, even changing the interpretation of the difficulty of the landscape for different executions. These observations support our claim that FDC (based on local optima) may not be a good metric to assess the difficulty of AutoML problems.

Figure 4.8: FDC of the regions explored by TPE and SMAC.



Source: author.

Chapter 5

Conclusions and Future Work

AutoML methods allow the industry and researchers from various fields to use machine learning without the need to deeply understand what happens behind the curtains. However, such methods still need to define a search space of possible solutions in order to find the best one for a given learning problem. Little is known about the characteristics of these spaces and how they can affect the performance of different search methods.

In this work, we adapted a tree-based representation of machine learning pipelines that is able to capture the semantics of the pipelines. We took advantage of this representation to define a probabilistic neighborhood based on a mutation operator and a distance metric between pipelines that take into consideration the semantics of the modifications needed to transform one pipeline into the other. Changes that are considered more relevant by a human expert (replacing a classification algorithm, for example) result in a larger distance than smaller modifications (changing the value of a hyperparameter, for example).

In order to address Research Question 1 (characteristics of the search space), we evaluated typical fitness landscape analysis measures for four different search spaces and nine datasets. We found that larger search spaces, especially those with broad domains for the hyperparameters of the algorithms, tend to have higher neutrality ratios than smaller spaces. Correlation Length, which is a way of measuring the ruggedness of a landscape, showed that smaller search spaces are usually more rugged than larger ones, which can be hard for optimization methods. Fitness Distance Correlation (FDC) results, on the other hand, were harder to analyze. FDC values show great variation between different search spaces and datasets, with no apparent patterns. Although results were difficult to analyze, more than 60% of the evaluated instances were considered easy, less than a third were considered misleading and very few are hard. This suggests that, although they seem very complex, given their size and variety of variable types, AutoML search spaces may not be extremely hard to search, at least for methods based on Bayesian optimization. Furthermore, we note that using local optima to calculate FDC can lead to very different results compared to experiments using global optima, which is usually unfeasible to calculate.

Regarding Research Question 2 (optimizer performance), search method TPE

found slightly better pipelines and explored a larger portion of the search space for more neutral landscapes, suggesting that it may benefit from the presence of neutrality, but this effect was less present in SMAC executions. More rugged landscapes, as identified by Correlation Length, often result in poorer exploration by both TPE and SMAC, although SMAC is more robust than TPE. However, SMAC was able to explore some very rugged landscapes better than smoother ones, suggesting that other characteristics of the landscape are affecting the exploration ability of the optimizers. We found no clear correlation between FDC values and the optimizers' ability to explore the landscapes and find good solutions, indicating that this metric may not be a good way of explaining the performance of AutoML methods based on Bayesian optimization.

These results give initial insights into the characteristics of AutoML search spaces and how search methods based on Bayesian optimization behave in relation to these characteristics. A possible direction for future research is to evaluate other classes of AutoML methods, such as those based on evolutionary search and multi-fidelity optimization. It would also be interesting to modify the grammars in order to eliminate the generation of different pipelines that are actually considered the same, because some hyperparameters are ignored during training. Another important direction is to study how the parameters that control the generation of neighborhoods (i.e., the weights of the mutation operator) and the distance between different components of the pipelines, which were defined based on the experience of the author and his advisors, affect FLA measures. The distance metric also depends on a categorization of machine learning algorithms. In this work, we used the classification used by the scikit-learn library, but there might be more appropriate approaches, such as grouping algorithms based on how they work [70].

Bibliography

[1] Wolfgang Banzhaf, Peter Nordin, Robert E Keller, and Frank D Francone. *Genetic Programming: An Introduction on the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann Publishers Inc., 1st edition, 1998.

[2] Lionel Barnett. Ruggedness and neutrality: The NKp family of fitness landscapes. In *Artificial Life VI: Proceedings of the Sixth International Conference on Artificial Life*, pages 18–27. MIT Press, 1998.

[3] Márcio P Basgalupp, Rodrigo C Barros, Alex GC de Sá, Gisele L Pappa, Rafael G Mantovani, André CPLF de Carvalho, and Alex A Freitas. An extensive experimental evaluation of automated machine learning methods for recommending classification algorithms. *Evolutionary Intelligence*, 14(4):1895–1914, 2021.

[4] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554. Curran Associates, Inc., 2011.

[5] James S Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning*, pages 115–123. PMLR, 2013.

[6] Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Frank Hutter, Michel Lang, Rafael G Mantovani, Jan N Van Rijn, and Joaquin Vanschoren. OpenML benchmarking suites and the OpenML100. *arXiv preprint arXiv:1708.03731*, 2017.

[7] Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Frank Hutter, Michel Lang, Rafael G Mantovani, Jan N van Rijn, and Joaquin Vanschoren. OpenML benchmarking suites. *arXiv:1708.03731v2 [stat.ML]*, 2019.

[8] Anna S Bosman, Andries P Engelbrecht, and Mardé Helbig. Search space boundaries in neural network error landscape analysis. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2016.

[9] Anna S Bosman, Andries P Engelbrecht, and Mardé Helbig. Fitness landscape analysis of weight-elimination neural networks. *Neural Processing Letters*, 48(1):353–373, 2018.

[10] Anna S Bosman, Andries P Engelbrecht, and Mardé Helbig. Progressive gradient walk for neural network fitness landscape analysis. In *Proceedings of the 2018 Genetic and Evolutionary Computation Conference*, pages 1473–1480. ACM, 2018.

[11] Hannelore Brandt. Correlation analysis of fitness landscapes. *IIASA Interim Report*, 2001.

[12] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Müller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake Vanderplas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238*, 2013.

[13] Scott Cost and Steven Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10(1):57–78, 1993.

[14] Carles M Cuadras. Distance analysis in discrimination and classification using both continuous and categorical variables. In *Statistical Data Analysis and Inference*, pages 459–473. Elsevier, 1989.

[15] Zbigniew J Czech. Statistical measures of a fitness landscape for the vehicle routing problem. In *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8. IEEE, 2008.

[16] Silvia N das Dôres, Carlos Soares, and Duncan Ruiz. Bandit-based automated machine learning. In *Proceedings of the 7th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 121–126. IEEE, 2018.

[17] Alex GC de Sá, Walter JGS Pinto, Luiz OVB Oliveira, and Gisele L Pappa. RECIPE: a grammar-based framework for automatically evolving classification pipelines. In *Proceedings of the European Conference on Genetic Programming (Part of EvoStar)*, pages 246–261. Springer, 2017.

[18] Alex GC de Sá, Cristiano G Pimenta, Gisele L Pappa, and Alex A Freitas. A robust experimental evaluation of automated multi-label classification methods. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 175–183. ACM, 2020.

[19] Bilel Derbel, Arnaud Liefooghe, Sébastien Vérel, Hernan Aguirre, and Kiyoshi Tanaka. New features for continuous exploratory landscape analysis based on the SOO tree. In *Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*, pages 72–86. ACM, 2019.

[20] Radwa Elshawi, Mohamed Maher, and Sherif Sakr. Automated machine learning: State-of-the-art and open challenges. *arXiv preprint arXiv:1906.02287*, 2019.

[21] Andries P Engelbrecht, P Bosman, and Katherine M Malan. The influence of fitness landscape characteristics on particle swarm optimisers. *Natural Computing*, 21(2): 335–345, 2022.

[22] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1437–1446. PMLR, 2018.

[23] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, pages 2962–2970. Curran Associates, Inc., 2015.

[24] Cyril Fonlupt, Denis Robilliard, and Philippe Preux. Fitness landscape and the behavior of heuristics. In *Evolution Artificielle*, pages 321–329, 1997.

[25] Unai Garciarena, Roberto Santana, and Alexander Mendiburu. Analysis of the complexity of the automatic pipeline generation problem. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2018.

[26] Nicholas Geard, Janet Wiles, Jennifer Hallinan, Bradley Tonkes, and Ben Skellett. A comparison of neutral landscapes-NK, NKp and NKq. In *2002 Congress on Evolutionary Computation (CEC)*, pages 205–210. IEEE, 2002.

[27] Kyle R Harrison, Beatrice M Ombuki-Berman, and Andries P Engelbrecht. The parameter configuration landscape: A case study on particle swarm optimization. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 808–814. IEEE, 2019.

[28] Frank Hutter. *Automated configuration of algorithms for solving hard computational problems*. PhD thesis, University of British Columbia, 2009.

[29] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

[30] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 1st edition, 2019.

[31] Eduardo Izquierdo-Torres. Evolving dynamical systems: nearly neutral regions in continuous fitness landscapes. Master's thesis, University of Sussex, 2004.

[32] Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, Inc., 1st edition, 1990.

[33] Donald R Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21(4):345–383, 2001.

[34] Terry Jones. *Evolutionary algorithms, fitness landscapes and search*. PhD thesis, The University of New Mexico, 1995.

[35] Terry Jones and Stephanie Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192. Morgan Kaufmann Publishers Inc., 1995.

[36] Stuart Kauffman and Simon Levin. Towards a general theory of adaptive walks on rugged landscapes. *Journal of Theoretical Biology*, 128(1):11–45, 1987.

[37] David J Kedziora, Katarzyna Musial, and Bogdan Gabrys. Autonoml: Towards an integrated framework for autonomous machine learning. *arXiv preprint arXiv:2012.12600*, 2020.

[38] Pascal Kerschke, Holger H Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3–45, 2019.

[39] Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. In *Data Mining and Constraint Programming*, pages 149–190. Springer, 2016.

[40] Ryan D Lang and Andries P Engelbrecht. Distributed random walks for fitness landscape analysis. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 612–619. ACM, 2020.

[41] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

[42] Wei Li, Bo Sun, Ying Huang, and Soroosh Mahmoodi. Adaptive complex network topology with fitness distance correlation framework for particle swarm optimization. *International Journal of Intelligent Systems*, 37(8), 2021.

[43] Maxwell W Libbrecht and William S Noble. Machine learning applications in genetics and genomics. *Nature Reviews Genetics*, 16(6):321–332, 2015.

[44] Katherine M Malan. A survey of advances in landscape analysis for optimisation. *Algorithms*, 14(2):40, 2021.

[45] Katherine M Malan and Andries P Engelbrecht. A survey of techniques for characterising fitness landscapes and some possible ways forward. *Information Sciences*, 241:148–163, 2013.

[46] Katherine M Malan and Andries P Engelbrecht. Characterising the searchability of continuous optimisation problems for PSO. *Swarm Intelligence*, 8(4):275–302, 2014.

[47] Katherine M Malan and Andries P Engelbrecht. A progressive random walk algorithm for sampling continuous fitness landscapes. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 2507–2514. IEEE, 2014.

[48] Brendan McCane and Michael Albert. Distance functions for categorical and mixed variables. *Pattern Recognition Letters*, 29(7):986–993, 2008.

[49] Robert I Mckay, Nguyen X Hoai, Peter A Whigham, Yin Shan, and Michael O'Neill. Grammar-based genetic programming: A survey. *Genetic Programming and Evolvable Machines*, 11(3–4):365–396, 2010.

[50] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. Exploratory landscape analysis. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pages 829–836. ACM, 2011.

[51] Tom M Mitchell. *Machine Learning*. McGraw Hill, 1st edition, 1997.

[52] Kevin P Murphy. *Machine Learning: A Probabilistic Perspective*. MIT press, 1st edition, 2012.

[53] Matheus Nunes, Paulo M Fraga, and Gisele L Pappa. Fitness landscape analysis of graph neural network architecture search spaces. In *Proceedings of the 2021 Genetic and Evolutionary Computation Conference*, pages 876–884. ACM, 2021.

[54] Gabriela Ochoa and Nadarajen Veerapen. Mapping the global structure of TSP fitness landscapes. *Journal of Heuristics*, 24(3):265–294, 2018.

[55] Gabriela Ochoa, Rong Qu, and Edmund K Burke. Analyzing the landscape of a graph based hyper-heuristic for timetabling problems. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, pages 341–348. ACM, 2009.

[56] Gabriela Ochoa, Sébastien Vérel, Fabio Daolio, and Marco Tomassini. Local optima networks: A new model of combinatorial fitness landscapes. In *Recent Advances in the Theory and Application of Fitness Landscapes*, pages 233–262. Springer, 2014.

[57] Randal S Olson and Jason H Moore. TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on Automatic Machine Learning*, pages 66–74. PMLR, 2016.

[58] Trilok N Pandey, Alok K Jagadev, Suman K Mohapatra, and Satchidananda Dehuri. Credit risk analysis using machine learning classifiers. In *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, pages 1850–1854. IEEE, 2017.

[59] Gisele L Pappa, Gabriela Ochoa, Matthew R Hyde, Alex A Freitas, John Woodward, and Jerry Swan. Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms. *Genetic Programming and Evolvable Machines*, 15(1):3–35, 2014.

[60] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplass, Alexandre Passos, and David Cournapeau. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[61] Cristiano G Pimenta, Alex GC de Sá, Gabriela Ochoa, and Gisele L Pappa. Fitness landscape analysis of automated machine learning search spaces. In *Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar)*, pages 114–130. Springer, 2020.

[62] Erik Pitzer and Michael Affenzeller. A comprehensive survey on fitness landscape analysis. In *Recent Advances in Intelligent Engineering Systems*, pages 161–191. Springer, 2012.

[63] Yasha Pushak. *Algorithm configuration landscapes: analysis and exploitation.* PhD thesis, University of British Columbia, 2022.

[64] Yasha Pushak and Holger Hoos. Algorithm configuration landscapes. In *International Conference on Parallel Problem Solving from Nature*, pages 271–283. Springer, 2018.

[65] Yasha Pushak and Holger Hoos. AutoML loss landscapes. *ACM Transactions on Evolutionary Learning*, 2(3):1–30, 2022.

[66] Yasha Pushak and Holger H Hoos. Golden parameter search: exploiting structure to quickly configure parameters in parallel. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 245–253. ACM, 2020.

[67] Anna Rakitianskaia, Eduan Bekker, Katherine M Malan, and Andries P Engelbrecht. Analysis of error landscapes in multi-layered neural networks for classification. In

2016 IEEE Congress on Evolutionary Computation (CEC), pages 5270–5277. IEEE, 2016.

[68] Christian M Reidys and Peter F Stadler. Neutrality in fitness landscapes. *Applied Mathematics and Computation*, 117(2-3):321–350, 2001.

[69] John R Rice. The algorithm selection problem. In *Advances in Computers*, volume 15, pages 65–118. Elsevier, 1976.

[70] Adriano Rivolli, Jesse Read, Carlos Soares, Bernhard Pfahringer, and André CPLF de Carvalho. An empirical analysis of binary transformation strategies and base algorithms for multi-label learning. *Machine Learning*, 109:1509–1563, 2020.

[71] Nuno M Rodrigues, Katherine M Malan, Gabriela Ochoa, Leonardo Vanneschi, and Sara Silva. Fitness landscape analysis of convolutional neural network architectures for image classification. *Information Sciences*, 609:711–726, 2022.

[72] Cullen Schaffer. A conservation law for generalization performance. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 259–265. Elsevier, 1994.

[73] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

[74] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 3rd edition, 2012.

[75] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information processing systems*. Curran Associates, Inc., 2012.

[76] Peter F Stadler. Fitness landscapes. In *Biological Evolution and Statistical Physics*, pages 183–204. Springer, 2002.

[77] Matheus C Teixeira and Gisele L Pappa. Understanding AutoML search spaces with local optima networks. In *Proceedings of the 2022 Genetic and Evolutionary Computation Conference*, pages 449–457. ACM, 2022.

[78] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 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*, pages 847–855. ACM, 2013.

[79] Kalifou R Traoré, Andrés Camero, and Xiao X Zhu. Fitness landscape footprint: A framework to compare neural architecture search problems. *arXiv preprint arXiv:2111.01584*, 2021.

[80] Kalifou R Traoré, Andrés Camero, and Xiao X Zhu. HPO: We won't get fooled again. *arXiv preprint arXiv:2208.03320*, 2022.

[81] Willem A van Aardt, Anna S Bosman, and Katherine M Malan. Characterising neutrality in neural network error landscapes. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1374–1381. IEEE, 2017.

[82] Leonardo Vanneschi, Yuri Pirola, and Philippe Collard. A quantitative study of neutrality in GP boolean landscapes. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pages 895–902. ACM, 2006.

[83] Leonardo Vanneschi, Yuri Pirola, Giancarlo Mauri, Marco Tomassini, Philippe Collard, and Sébastien Vérel. A study of the neutrality of boolean function landscapes in genetic programming. *Theoretical Computer Science*, 425:34–57, 2012.

[84] Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15 (2):49–60, 2013.

[85] Sébastien Vérel, Gabriela Ochoa, and Marco Tomassini. Local optima networks of NK landscapes with neutrality. *IEEE Transactions on Evolutionary Computation*, 15(6):783–797, 2010.

[86] Edward Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63(5):325–336, 1990.

[87] Randall Wilson and Tony R Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997.

[88] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., 4th edition, 2016.

[89] David H Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.

[90] David H Wolpert and William G Macready. No free lunch theorems for search. Technical report, Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

[91] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[92] Sewall Wright. The roles of mutation, inbreeding, crossbreeding, and selection in evolution. In *Proceedings of the 6th International Congress on Genetics*, pages 356–366. Brooklyn Botanic Garden, 1932.

[93] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.

[94] Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*, 2018.

[95] Marc-André Zöller and Marco F Huber. Survey on automated machine learning. *Journal of Artificial Intelligence Research*, 70:409–472, 2021.

Appendix A

Context-free Grammars

This appendix lists the grammars presented in Section 3.1 in BNF (Backus–Naur form) notation.

A.1 Search space `small`

```
# __default denotes the default value of the parameter,
# as defined by the scikit-learn implementation.

<Start> ::= <preprocessing> <classification>

# Preprocessing
<preprocessing> ::=
    <imputation> | <imputation> <binarizer> | imputation> <bounding> |
    <imputation> <dimensionality> | <imputation> <feat_const> |
    <imputation> <binarizer> <dimensionality> |
    <imputation> <bounding> <binarizer> |
    <imputation> <bounding> <dimensionality> |
    <imputation> <dimensionality> <binarizer> |
    <imputation> <dimensionality> <bounding> |
    <imputation> <dimensionality> <feat_const> |
    <imputation> <feat_const> <binarizer> |
    <imputation> <feat_const> <bounding> |
    <imputation> <feat_const> <dimensionality> |
    <imputation> <bounding> <binarizer> <dimensionality> |
    <imputation> <bounding> <dimensionality> <binarizer> |
    <imputation> <dimensionality> <bounding> <binarizer> |
    <imputation> <dimensionality> <feat_const> <binarizer> |
    <imputation> <dimensionality> <feat_const> <bounding> |
    <imputation> <feat_const> <binarizer> <dimensionality> |
    <imputation> <feat_const> <bounding> <binarizer> |
    <imputation> <feat_const> <bounding> <dimensionality> |
```

```
    <imputation> <feat_const> <dimensionality> <binarizer> |
    <imputation> <feat_const> <dimensionality><bounding>

## Imputation
<imputation> ::= <simpleImputer>

<simpleImputer> ::= SimpleImputer <strategy_imp>
<strategy_imp> ::= __default

## Bounding
<bounding> ::=
    <normalizer> | <minMaxScaler> | <maxAbsScaler> | <robust_scaler> |
    <standard_scaler> | <quantile_transformer>

<normalizer> ::= Normalizer <norm>
<norm> ::= __default

<minMaxScaler> ::= MinMaxScaler

<maxAbsScaler> ::= MaxAbsScaler

<robust_scaler> ::= RobustScaler <with_scaling> <with_centering>
<with_scaling> ::= __default
<with_centering> ::= __default

<standard_scaler> ::= StandardScaler <with_std> <with_mean>
<with_std> ::= __default
<with_mean> ::= __default

<quantile_transformer> ::=
    QuantileTransformer <n_quantiles> <output_distribution>
<n_quantiles> ::= __default
<output_distribution> ::= __default

## Binarizer
<binarizer> ::= <binarizer_alg>

<binarizer_alg> ::= Binarizer <threshold_bin>
<threshold_bin> ::= __default

## Dimensionality reduction and feature selection
<dimensionality> ::=
    <varianceThreshold> | <selectKBest> | <univariate_select> |
    <select_percentile> | <pca> | <incremental_pca> | <kernel_pca> |
    <fast_ica> | <gaussian_projection> | <sparse_random_projection> |
    <feature_agglomeration> | <rbf_sampler> | <nystroem> |
    <truncatedsvd> | <linear_svc_dim> | <extra_trees_dim>
```

```
<varianceThreshold> ::= VarianceThreshold

<selectKBest> ::= SelectKBest <features_dim>
<features_dim> ::= __default

<univariate_select> ::=
    UnivariateSelect <score_func_univar> <mode> <param_univar>
<score_func_univar> ::= __default
<mode> ::= __default
<param_univar> ::= __default

<select_percentile> ::= SelectPercentile <score_func_select> <percentile>
<score_func_select> ::= __default
<percentile> ::= __default

<pca> ::=
    PCA <features_dim> <whiten> <svd_solver> <tol_pca> <iterated_power>
<whiten> ::= __default
<svd_solver> ::= __default
<tol_pca> ::= __default
<iterated_power> ::= __default

<incremental_pca> ::= IncrementalPCA <features_dim> <whiten>

<kernel_pca> ::=
    KernelPCA <features_dim> <kernel_pca_hp> <degree_pca>
              <gamma_dim> <coef0_dim>
<kernel_pca_hp> ::= __default
<degree_pca> ::= __default
<gamma_dim> ::= __default
<coef0_dim> ::= __default

<fast_ica> ::=
    FastICA <features_dim> <algorithm_fastica> <funct>
            <max_iter_fastica> <tol_dim> <whiten>
<algorithm_fastica> ::= __default
<funct> ::= __default
<max_iter_fastica> ::= __default
<tol_dim> ::= __default

<gaussian_projection> ::= GaussianRandomProjection <features_dim> <epsilon>
<epsilon> ::= __default

<sparse_random_projection> ::=
    SparseRandomProjection <features_dim> <epsilon>
                           <density> <dense_output>
```

```
<density> ::= __default
<dense_output> ::= __default

<feature_agglomeration> ::=
    FeatureAgglomeration <features_dim> <affinity> <compute_full_tree>
<affinity> ::= __default __default
<compute_full_tree> ::= __default

<rbf_sampler> ::= RBFSampler <features_dim> <gamma_dim>

<nystroem> ::=
    Nystroem <features_dim> <kernel_dr> <gamma_dim> <degree_1> <coef0_dim>
<kernel_dr> ::= __default
<degree_1> ::= __default

<truncatedsvd> ::=
    TruncatedSVD <features_dim> <n_iter> <tol_dim> <algorithm_tsvd>
<n_iter> ::= __default
<algorithm_tsvd> ::= __default

<linear_svc_dim> ::= LinearSVCPreprocessing <tol_svc_dim> <C_dim>
<tol_svc_dim> ::= __default
<C_dim> ::= __default

<extra_trees_dim> ::=
    ExtraTreesPreprocessing <criterion_dim> <max_features_dim>
                            <min_samples_split_dim> <min_samples_leaf_dim>
                            <bootstrap_dim>
<criterion_dim> ::= __default
<max_features_dim> ::= __default
<min_samples_split_dim> ::= __default
<min_samples_leaf_dim> ::= __default
<bootstrap_dim> ::= __default

## Feature construction
<feat_const> ::=
    <polynominal_features> | <bernoulli_rbm> | <random_trees_emb>

<polynominal_features> ::=
    PolynomialFeatures <degree_polyfeat> <interaction_only> <include_bias>
<degree_polyfeat> ::= __default
<interaction_only> ::= __default
<include_bias> ::= __default

<bernoulli_rbm> ::=
    BernoulliRBM <n_components> <learning_rate_brbm>
                 <batch_size> <max_iter_dim>
```

```
<n_components> ::= __default
<learning_rate_brbm> ::= __default
<batch_size> ::= __default
<max_iter_dim> ::= __default

<random_trees_emb> ::=
    RandomTreesEmbedding <n_estimators_feat> <max_depth_feat>
                         <min_samples_split_feat> <min_samples_leaf_feat>
<n_estimators_feat> ::= __default
<max_depth_feat> ::= __default
<min_samples_split_feat> ::= __default
<min_samples_leaf_feat> ::= __default

# Classification
<classification> ::=
    <gaussian_nb> | <bernoulli_nb> | <multinomial_nb> | <complement_nb> |
    <svc> | <nu_svc> | <linear_svc> | <logistic_regression> |
    <perceptron> | <passive_agressive> | <mlp> | <sgd> | <lda> | <qda> |
    <knn> | <radius_neighbours> | <centroid> | <ridge> | <ridge_cv> |
    <decision_tree> | <extra_tree> | <random_forest> |
    <extra_trees> | <ada_boost> | <gradient_boosting>

<gaussian_nb> ::= GaussianNB

<bernoulli_nb> ::= BernoulliNB <binarize> <alpha_nb> <fit_prior>
<binarize> ::= __default
<alpha_nb> ::= __default
<fit_prior> ::= __default

<multinomial_nb> ::= MultinomialNB <alpha_nb> <fit_prior>

<complement_nb> ::= ComplementNB <alpha_nb> <fit_prior> <norm_cnb>
<norm_cnb> ::= __default

<svc> ::=
    SVC <C> <kernel> <degree_kernel> <gamma> <coef0> <probability>
        <shrinking> <decision_function_shape> <tol> <max_iter>
        <class_weight>
<C> ::= __default
<kernel> ::= __default
<degree_kernel> ::= __default
<gamma> ::= __default
<coef0> ::= __default
<probability> ::= __default
<shrinking> ::= __default
<decision_function_shape> ::= __default
<tol> ::= __default
```

```
<max_iter> ::= __default
<class_weight> ::= __default

<nu_svc> ::=
    NuSVC <nu> <kernel> <degree_kernel> <gamma> <coef0> <probability>
          <shrinking> <decision_function_shape> <tol> <max_iter>
          <class_weight>
<nu> ::= __default

<linear_svc> ::=
    LinearSVC <penalty_loss_dual_svc> <tol> <C> <class_weight>
<penalty_loss_dual_svc> ::= __default __default __default

<logistic_regression> ::=
    LogisticRegression <penalty> <tol> <C> <fit_intercept>
                       <max_iter> <warm_start>
<penalty> ::= __default
<fit_intercept> ::= __default
<warm_start> ::= __default

<perceptron> ::= Perceptron <penalty> <tol> <max_iter> <warm_start>

<passive_agressive> ::=
    PassiveAggressive <C> <fit_intercept> <max_iter> <tol>
                      <loss_pac> <average> <class_weight>
<loss_pac> ::= __default
<average> ::= __default

<mlp> ::=
    MLP <learning_rate> <learning_rate_init> <momentum>
        <max_iter> <activation>
<learning_rate> ::= __default
<learning_rate_init> ::= __default
<momentum> ::= __default
<activation> ::= __default

<sgd> ::= SGD <penalty> <tol> <max_iter> <loss> <warm_start>
<loss> ::= __default

<lda> ::= LDA <tol>

<qda> ::= QDA <reg_param> <tol>
<reg_param> ::= __default

<knn> ::=
    KNearestNeighbors <k> <weights> <k_algorithm>
                      <leaf_size> <p> <d_metric>
```

```
<k> ::= __default
<weights> ::= __default
<k_algorithm> ::= __default
<leaf_size> ::= __default
<p> ::= __default
<d_metric> ::= __default

<radius_neighbours> ::=
    RadiusNeighbors <radius> <weights> <k_algorithm>
                    <leaf_size> <p> <d_metric>
<radius> ::= __default

<centroid> ::= Centroid <shrinking_threshold> <d_metric>
<shrinking_threshold> ::= __default

<ridge> ::=
    Ridge <alpha> <max_iter> <copy_X> <solver_ridge> <tol>
          <normalize> <fit_intercept>
<alpha> ::= __default
<copy_X> ::= __default
<solver_ridge> ::= __default
<normalize> ::= __default

<ridge_cv> ::= RidgeCCV <cv> <normalize> <fit_intercept>
<cv> ::= __default

<decision_tree> ::=
    DT <criterion> <splitter> <max_depth> <max_features>
       <min_weight_fraction_leaf> <max_leaf_nodes>
<criterion> ::= __default
<splitter> ::= __default
<max_depth> ::= __default
<max_features> ::= __default
<min_weight_fraction_leaf> ::= __default
<max_leaf_nodes> ::= __default

<extra_tree> ::=
    ExtraTree <criterion> <splitter> <class_weight> <max_features>
              <max_depth> <min_weight_fraction_leaf> <max_leaf_nodes>

<random_forest> ::=
    RandomForest <criterion> <bootstrap_and_oob> <class_weight_Trees>
                 <n_estimators> <warm_start> <max_features> <max_depth>
                 <min_weight_fraction_leaf> <max_leaf_nodes>
<bootstrap_and_oob> ::= __default __default
<class_weight_Trees> ::= __default
<n_estimators> ::= __default
```

```
<extra_trees> ::=
    ExtraTrees <criterion> <bootstrap_and_oob> <class_weight_Trees>
               <n_estimators> <warm_start> <max_features> <max_depth>
               <min_weight_fraction_leaf> <max_leaf_nodes>

<ada_boost> ::= AdaBoost <algorithm_ada> <n_estimators> <learning_rate_ada>
<algorithm_ada> ::= __default
<learning_rate_ada> ::= __default

<gradient_boosting> ::=
    GradientBoosting <loss_gradient> <tol> <learning_rate_gradient>
                     <presort> <n_estimators> <warm_start> <max_features>
                     <max_depth> <min_weight_fraction_leaf>
                     <max_leaf_nodes>
<loss_gradient> ::= __default
<learning_rate_gradient> ::= __default
<presort> ::= __default
```

A.2 Search space medium

```
<Start> ::= <preprocessing> <classification>

# Preprocessing
<preprocessing> ::=
    <imputation> | <imputation> <bounding> | <imputation> <dimensionality>

## Imputation
<imputation> ::= <simpleImputer>

<simpleImputer> ::= SimpleImputer <strategy_imp>
<strategy_imp> ::= mean | median | most_frequent

## Bounding
<bounding> ::= <standard_scaler>

<standard_scaler> ::= StandardScaler <with_std> <with_mean>
<with_std> ::= True | False
<with_mean> ::= True | False

## Dimensionality reduction and feature selection
<dimensionality> ::= <selectKBest> | <pca>

<selectKBest> ::= SelectKBest <features_dim>
<features_dim> ::= RANDATT(1,ATT-1)

<pca> ::= PCA <features_dim> <whiten> <svd_solver>
<whiten> ::= True | False
<svd_solver> ::= full | arpack | randomized

# Classification
<classification> ::=
    <logistic_regression> | <mlp> | <knn> | <random_forest> | <ada_boost>

<logistic_regression> ::=
    LogisticRegression <penalty> <fit_intercept> <max_iter> <warm_start>
<penalty> ::= l1 | l2
<fit_intercept> ::= True | False
<max_iter>  ::= 100 | 300 | 500
<warm_start> ::= True | False

<mlp> ::= MLP <learning_rate> <max_iter> <activation>
<learning_rate> ::= constant | invscaling | adaptive
<activation> ::= identity | logistic | tanh | relu
```

```
<knn> ::=
    KNearestNeighbors <k> <weights> <k_algorithm>
                      <leaf_size> <p> <d_metric>
<k> ::= 1 | 5 | 15 | 25 | 35
<weights> ::= uniform | distance
<k_algorithm> ::= brute | kd_tree | ball_tree
<leaf_size> ::= 20 | 40 | 60 | 80 | 100
<p> ::= RANDINT(1,10)
<d_metric> ::= euclidean | manhattan | chebyshev | minkowski

<random_forest> ::=
    RandomForest <criterion> <bootstrap_and_oob> <class_weight_Trees>
                 <n_estimators> <warm_start> <max_features> <max_depth>
<criterion> ::= gini | entropy
<bootstrap_and_oob> ::= True True | True False | False False
<class_weight_Trees> ::= balanced | balanced_subsample | None
<n_estimators> ::= 10 | 30 | 50
<max_features> ::= sqrt | log2
<max_depth> ::= 10 | 30 | 50

<ada_boost> ::= AdaBoost <algorithm_ada> <n_estimators>
<algorithm_ada> ::= SAMME.R | SAMME
```

A.3 Search space large

```
<Start> ::= <preprocessing> <classification>

# Preprocessing
<preprocessing> ::=
    <imputation> | <imputation> <binarizer> | <imputation> <bounding> |
    <imputation> <dimensionality> | <imputation> <feat_const> |
    <imputation> <binarizer> <dimensionality> |
    <imputation> <bounding> <binarizer> |
    <imputation> <bounding> <dimensionality> |
    <imputation> <dimensionality> <binarizer> |
    <imputation> <dimensionality> <bounding> |
    <imputation> <dimensionality> <feat_const> |
    <imputation> <feat_const> <binarizer> |
    <imputation> <feat_const> <bounding> |
    <imputation> <feat_const> <dimensionality> |
    <imputation> <bounding> <binarizer> <dimensionality> |
    <imputation> <bounding> <dimensionality> <binarizer> |
    <imputation> <dimensionality> <bounding> <binarizer> |
    <imputation> <dimensionality> <feat_const> <binarizer> |
    <imputation> <dimensionality> <feat_const> <bounding> |
    <imputation> <feat_const> <binarizer> <dimensionality> |
    <imputation> <feat_const> <bounding> <binarizer> |
    <imputation> <feat_const> <bounding> <dimensionality> |
    <imputation> <feat_const> <dimensionality> <binarizer> |
    <imputation> <feat_const> <dimensionality><bounding>

## Imputation
<imputation> ::= <simpleImputer>

<simpleImputer> ::= SimpleImputer <strategy_imp>
<strategy_imp> ::= mean | median | most_frequent

## Bounding
<bounding> ::=
    <normalizer> | <minMaxScaler> | <maxAbsScaler> |
    <robust_scaler> | <standard_scaler> | <quantile_transformer>

<normalizer> ::= Normalizer <norm>
<norm> ::= l1 | l2 | max

<minMaxScaler> ::= MinMaxScaler

<maxAbsScaler> ::= MaxAbsScaler
```

```
<robust_scaler> ::= RobustScaler <with_scaling> <with_centering>
<with_scaling> ::= True | False
<with_centering> ::= True | False

<standard_scaler> ::= StandardScaler <with_std> <with_mean>
<with_std> ::= True | False
<with_mean> ::= True | False

<quantile_transformer> ::=
    QuantileTransformer <n_quantiles> <output_distribution>
<n_quantiles> ::= RANDINT(10,2000)
<output_distribution> ::= uniform | normal

## Binarizer
<binarizer> ::= <binarizer_alg>

<binarizer_alg> ::= Binarizer <threshold_bin>
<threshold_bin> ::= RANDFLOAT(0.000001,1000)

## Dimensionality reduction and feature selection
<dimensionality> ::=
    <varianceThreshold> | <selectKBest> | <univariate_select> |
    <select_percentile> | <pca> | <incremental_pca> | <kernel_pca> |
    <fast_ica> | <gaussian_projection> | <sparse_random_projection> |
    <feature_agglomeration> | <rbf_sampler> | <nystroem> |
    <truncatedsvd> | <linear_svc_dim> | <extra_trees_dim>

<varianceThreshold> ::= VarianceThreshold

<selectKBest> ::= SelectKBest <features_dim>
<features_dim> ::= RANDATT(1,ATT-1)

<univariate_select> ::=
    UnivariateSelect <score_func_univar> <mode> <param_univar>
<score_func_univar> ::= chi2 | f_classif
<mode> ::= fpr | fdr | fwe
<param_univar> ::= RANDFLOAT(0.01,0.5)

<select_percentile> ::= SelectPercentile <score_func_select> <percentile>
<score_func_select> ::= chi2 | f_classif | mutual_info
<percentile> ::= RANDINT(1,99)

<pca> ::=
    PCA <features_dim> <whiten> <svd_solver> <tol_pca> <iterated_power>
<whiten> ::= True | False
<svd_solver> ::= auto | full | arpack | randomized
```

```
<tol_pca> ::= RANDFLOAT(0.0,0.1)
<iterated_power> ::= RANDINT(2,20)

<incremental_pca> ::= IncrementalPCA <features_dim> <whiten>

<kernel_pca> ::=
    KernelPCA <features_dim> <kernel_pca_hp> <degree_pca>
              <gamma_dim> <coef0_dim>
<kernel_pca_hp> ::= poly | rbf | sigmoid | cosine
<degree_pca> ::= RANDINT(2,5)
<gamma_dim> ::= RANDFLOAT(0.000030518,8.0)
<coef0_dim> ::= RANDFLOAT(-1,1)

<fast_ica> ::=
    FastICA <features_dim> <algorithm_fastica> <funct>
            <max_iter_fastica> <tol_dim> <whiten>
<algorithm_fastica> ::= parallel | deflation
<funct> ::= logcosh | exp | cube
<max_iter_fastica> ::= RANDINT(10,1000)
<tol_dim> ::= RANDFLOAT(0.0000000001,0.1)

<gaussian_projection> ::= GaussianRandomProjection <features_dim> <epsilon>
<epsilon> ::= RANDFLOAT(0.0,1.0)

<sparse_random_projection> ::=
    SparseRandomProjection <features_dim> <epsilon>
                           <density> <dense_output>
<density> ::= RANDFLOAT(0.00001,1.0)
<dense_output> ::= True | False

<feature_agglomeration> ::=
    FeatureAgglomeration <features_dim> <affinity>  <compute_full_tree>
<affinity> ::=
    euclidean ward | euclidean complete | euclidean average | l1 complete|
    l1 average | l2 complete | l2 average | manhattan complete |
    manhattan average | cosine complete | cosine average
<compute_full_tree> ::= True | False

<rbf_sampler> ::= RBFSampler <features_dim> <gamma_dim>

<nystroem> ::=
    Nystroem <features_dim> <kernel_dr> <gamma_dim> <degree_1> <coef0_dim>
<kernel_dr> ::= linear | poly | rbf | sigmoid
<degree_1> ::= RANDINT(2,10)

<truncatedsvd> ::=
    TruncatedSVD <features_dim> <n_iter> <tol_dim> <algorithm_tsvd>
```

```
<n_iter> ::= RANDINT(5,1000)
<algorithm_tsvd> ::= arpack | randomized

<linear_svc_dim> ::= LinearSVCPreprocessing <tol_svc_dim> <C_dim>
<tol_svc_dim> ::= RANDFLOAT(0.00001,0.1)
<C_dim> ::= RANDFLOAT(0.03125,32768.0)

<extra_trees_dim> ::=
    ExtraTreesPreprocessing <criterion_dim> <max_features_dim>
                            <min_samples_split_dim> <min_samples_leaf_dim>
                            <bootstrap_dim>
<criterion_dim> ::= gini | entropy
<max_features_dim> ::= RANDFLOAT(0.0,1.0)
<min_samples_split_dim> ::= RANDINT(2,20)
<min_samples_leaf_dim> ::= RANDINT(1,20)
<bootstrap_dim> ::= True | False

## Feature construction
<feat_const> ::=
    <polynominal_features> | <bernoulli_rbm> | <random_trees_emb>

<polynominal_features> ::=
    PolynomialFeatures <degree_polyfeat> <interaction_only> <include_bias>
<degree_polyfeat> ::= RANDINT(2,3)
<interaction_only> ::= True | False
<include_bias> ::= True | False

<bernoulli_rbm> ::=
    BernoulliRBM <n_components> <learning_rate_brbm>
                 <batch_size> <max_iter_dim>
<n_components> ::= RANDINT(1,10)
<learning_rate_brbm> ::= RANDFLOAT(0.01,1)
<batch_size> ::= RANDINT(1,100)
<max_iter_dim>  ::= RANDINT(10,10000)

<random_trees_emb> ::=
    RandomTreesEmbedding <n_estimators_feat> <max_depth_feat>
                         <min_samples_split_feat> <min_samples_leaf_feat>
<n_estimators_feat> ::= RANDINT(5,50)
<max_depth_feat> ::= RANDINT(2,10)
<min_samples_split_feat> ::= RANDINT(2,20)
<min_samples_leaf_feat> ::= RANDINT(1,20)

# Classification
<classification> ::=
    <gaussian_nb> | <bernoulli_nb> | <multinomial_nb> | <complement_nb> |
    <svc> | <nu_svc> | <linear_svc> | <logistic_regression> |
```

```
    <perceptron> | <passive_agressive> | <mlp> | <sgd> | <lda> | <qda> |
    <knn> | <radius_neighbours> | <centroid> | <ridge> | <ridge_cv> |
    <decision_tree> | <extra_tree> | <random_forest> | <extra_trees> |
    <ada_boost> | <gradient_boosting>

<gaussian_nb> ::= GaussianNB

<bernoulli_nb> ::= BernoulliNB <binarize> <alpha_nb> <fit_prior>
<binarize> ::= RANDFLOAT(0.0,1.0)
<alpha_nb> ::= RANDFLOAT(0.0,9.0)
<fit_prior> ::= True | False

<multinomial_nb> ::= MultinomialNB <alpha_nb> <fit_prior>

<complement_nb> ::= ComplementNB <alpha_nb> <fit_prior> <norm_cnb>
<norm_cnb> ::= True | False

<svc> ::=
    SVC <C> <kernel> <degree_kernel> <gamma> <coef0> <probability>
        <shrinking> <decision_function_shape> <tol> <max_iter>
        <class_weight>
<C> ::= RANDFLOAT(0.03125,32768.0)
<kernel> ::= linear | poly | rbf | sigmoid
<degree_kernel> ::= RANDINT(2,10)
<gamma> ::= RANDFLOAT(0.000030518,8.0)
<coef0> ::= RANDFLOAT(0.0,1000.0)
<probability> ::= True | False
<shrinking> ::= True | False
<decision_function_shape> ::= ovo | ovr | None
<tol> ::= RANDFLOAT(0.0000000001,0.1)
<max_iter> ::= RANDINT(10,10000)
<class_weight> ::= balanced | None

<nu_svc> ::=
    NuSVC <nu> <kernel> <degree_kernel> <gamma> <coef0> <probability>
          <shrinking> <decision_function_shape> <tol> <max_iter>
          <class_weight>
<nu> ::= RANDFLOAT(0.0000000001, 1.0)

<linear_svc> ::=
    LinearSVC <penalty_loss_dual_svc> <tol> <C> <class_weight>
<penalty_loss_dual_svc> ::=
    l1 squared_hinge False | l2 hinge True |
    l2 squared_hinge True | l2 squared_hinge False

<logistic_regression> ::=
    LogisticRegression <penalty> <tol> <C> <fit_intercept>
```

```
                        <max_iter> <warm_start>
<penalty> ::= l1 | l2
<fit_intercept> ::= True | False
<warm_start> ::= True | False

<perceptron> ::= Perceptron <penalty> <tol> <max_iter> <warm_start>

<passive_agressive> ::=
    PassiveAggressive <C> <fit_intercept> <max_iter> <tol>
                      <loss_pac> <average> <class_weight>
<loss_pac> ::= hinge | squared_hinge
<average> ::= True | False

<mlp> ::=
    MLP <learning_rate> <learning_rate_init> <momentum>
        <max_iter> <activation>
<learning_rate> ::= constant | invscaling | adaptive
<learning_rate_init> ::= RANDFLOAT(0.1,1.0)
<momentum> ::= RANDFLOAT(0.0,1.0)
<activation> ::= identity | logistic | tanh | relu

<sgd> ::= SGD <penalty> <tol> <max_iter> <loss> <warm_start>
<loss> ::=
    hinge | log | modified_huber | squared_hinge |
    perceptron | squared_loss

<lda> ::= LDA <tol>

<qda> ::= QDA <reg_param> <tol>
<reg_param> ::= RANDFLOAT(0.0,1.0)

<knn> ::=
    KNearestNeighbors <k> <weights> <k_algorithm>
                      <leaf_size> <p> <d_metric>
<k> ::= RANDINT(1,30)
<weights> ::= uniform | distance
<k_algorithm> ::= auto | brute | kd_tree | ball_tree
<leaf_size> ::= RANDINT(5,100)
<p> ::= RANDINT(1,15)
<d_metric> ::= euclidean | manhattan | chebyshev | minkowski

<radius_neighbours> ::=
    RadiusNeighbors <radius> <weights> <k_algorithm>
                    <leaf_size> <p> <d_metric>
<radius> ::= RANDFLOAT(1.0,30.0)

<centroid> ::= Centroid <shrinking_threshold> <d_metric>
```

```
<shrinking_threshold> ::= RANDFLOAT(0.0, 30.0)

<ridge> ::=
    Ridge <alpha> <max_iter> <copy_X> <solver_ridge>
          <tol> <normalize> <fit_intercept>
<alpha> ::= RANDFLOAT(0.0,1.0)
<copy_X> ::= True | False
<solver_ridge> ::=
    auto | svd | cholesky | lsqr | sparse_cg | sag | saga
<normalize> ::= True | False

<ridge_cv> ::= RidgeCCV <cv> <normalize> <fit_intercept>
<cv> ::= RANDINT(2,10)

<decision_tree> ::=
    DT <criterion> <splitter> <max_depth> <max_features>
       <min_weight_fraction_leaf> <max_leaf_nodes>
<criterion> ::= gini | entropy
<splitter> ::= best | random
<max_depth> ::= RANDINT(10,100)
<max_features> ::= sqrt | log2
<min_weight_fraction_leaf> ::= RANDFLOAT(0.0,0.5)
<max_leaf_nodes> ::= RANDINT(2,100)

<extra_tree> ::=
    ExtraTree <criterion> <splitter> <class_weight> <max_features>
              <max_depth> <min_weight_fraction_leaf> <max_leaf_nodes>

<random_forest> ::=
    RandomForest <criterion> <bootstrap_and_oob> <class_weight_Trees>
                 <n_estimators> <warm_start> <max_features> <max_depth>
                 <min_weight_fraction_leaf> <max_leaf_nodes>
<bootstrap_and_oob> ::= True True | True False | False False
<class_weight_Trees> ::= balanced | balanced_subsample | None
<n_estimators> ::= RANDINT(5,50)

<extra_trees> ::=
    ExtraTrees <criterion> <bootstrap_and_oob> <class_weight_Trees>
               <n_estimators> <warm_start> <max_features> <max_depth>
               <min_weight_fraction_leaf> <max_leaf_nodes>

<ada_boost> ::=
    AdaBoost <algorithm_ada> <n_estimators> <learning_rate_ada>
<algorithm_ada> ::= SAMME.R | SAMME
<learning_rate_ada> ::= RANDFLOAT(0.01,2.0)

<gradient_boosting> ::=
```

```
    GradientBoosting <loss_gradient> <tol> <learning_rate_gradient>
                     <presort> <n_estimators> <warm_start> <max_features>
                     <max_depth> <min_weight_fraction_leaf>
                     <max_leaf_nodes>
<loss_gradient> ::= deviance | exponential
<learning_rate_gradient> ::= RANDFLOAT(0.0000000001,1.0)
<presort> ::= True | False | auto
```

A.4 Search space auto-sklearn

```
<Start> ::= <preprocessing> <classification>

# Preprocessing
<preprocessing> ::=
    <imputation> | <imputation> <bounding> |
    <imputation> <dimensionality> | <imputation> <feat_const>

## Imputation
<imputation> ::= <simpleImputer>

<simpleImputer> ::= SimpleImputer

## Bounding
<bounding> ::= <normalizer> | <minMaxScaler> | <standard_scaler>

<normalizer> ::= Normalizer <norm>
<norm> ::= l1 | l2

<minMaxScaler> ::= MinMaxScaler <feature_range_min> <feature_range_max>
<feature_range_min> ::= RANDFLOAT(-1.0,0.0)
<feature_range_max> ::= RANDFLOAT(1.0,1.0)

<standard_scaler> ::= StandardScaler <with_std> <with_mean>
<with_std> ::= True | False
<with_mean> ::= True | False

## Dimensionality reduction
<dimensionality> ::= <pca>

<pca> ::= PCA <features_dim> <whiten>
<features_dim> ::= RANDATT(1,ATT-1)
<whiten> ::= True | False

## Feature construction
<feat_const> ::= <bernoulli_rbm>

<bernoulli_rbm> ::=
    BernoulliRBM <n_components> <learning_rate_brbm>
                 <batch_size> <max_iter_dim>
<n_components> ::= RANDINT(1,50)
<learning_rate_brbm> ::= RANDFLOAT(0.0001,1)
<batch_size> ::= RANDINT(1,100)
<max_iter_dim>  ::= RANDINT(1,1000)
```

```
# Classification
<classification> ::=
    <multinomial_nb> | <svc> | <linear_svc> | <sgd> |
    <knn> | <random_forest> | <extra_trees>

<multinomial_nb> ::= MultinomialNB <alpha_nb> <fit_prior>
<alpha_nb> ::= RANDFLOAT(0.0,1.0)
<fit_prior> ::= True | False

<svc> ::=
    SVC <kernel> <gamma> <degree_kernel> <max_iter>
        <C> <tol> <coef0> <shrinking>
<kernel> ::= linear | poly | rbf | sigmoid
<gamma> ::= RANDFLOAT(0.000030518,8.0)
<degree_kernel> ::= RANDINT(2,6)
<max_iter> ::= RANDINT(10000000,1000000000)
<C> ::= RANDFLOAT(0.00001,100000)
<tol> ::= RANDFLOAT(0.00001,0.01)
<coef0> ::= RANDFLOAT(0.0,10.0)
<shrinking> ::= True | False

<linear_svc> ::=
    LinearSVC <C> <tol> <multiclass> <intercept_scaling> <class_weight>
#<C> ::= RANDFLOAT(0.00001,100000)
#<tol> ::= RANDFLOAT(0.00001,0.01)
<multiclass> ::= ovr | crammer_singer
<intercept_scaling> ::= RANDFLOAT(0.1,10.0)
<class_weight> ::= balanced | None

<sgd> ::=
    SGD <loss> <penalty> <alpha_sgd> <l1_ratio> <tol> <max_iter>
        <learning_rate_sgd> <eta0> <power_t> <class_weight>
<loss> ::=
    hinge | log | modified_huber | squared_hinge | perceptron |
    squared_loss | huber | epsilon_insensitive |
    squared_epsilon_insensitive
<penalty> ::= l1 | l2 | elasticnet
<alpha_sgd> ::= RANDFLOAT(0.000001,0.1)
<l1_ratio> ::= RANDFLOAT(0.0,1.0)
#<tol> ::= RANDFLOAT(0.0000000001,0.1)
#<max_iter> ::= RANDINT(10000000,1000000000)
<learning_rate_sgd> ::= optimal | invscaling | constant
<eta0> ::= RANDFLOAT(0.00001,0.1)
<power_t> ::= RANDFLOAT(0.0,1.0)
#<class_weight> ::= balanced | None
```

```
<knn> ::= KNearestNeighbors <k> <weights>
<k> ::= RANDINT(1,50)
<weights> ::= uniform | distance

<random_forest> ::=
    RandomForest <n_estimators> <criterion> <max_features>
                 <max_depth> <min_samples_leaf> <bootstrap>
<n_estimators> ::= RANDINT(10,3000)
<criterion> ::= gini | entropy
<max_features> ::= sqrt | log2 | None | RANDFLOAT(0.0,1.0)
<max_depth> ::= None | RANDINT(2,4)
<min_samples_leaf> ::= RANDINT(1,50)
<bootstrap> ::= True | False

<extra_trees> ::=
    ExtraTrees <n_estimators> <criterion> <max_features>
               <max_depth> <min_samples_leaf> <bootstrap>
#<n_estimators> ::= RANDINT(10,3000)
#<criterion> ::= gini | entropy
#<max_features> ::= sqrt | log2 | None | RANDFLOAT(0.0,1.0)
#<max_depth> ::= None | RANDINT(2,4)
#<min_samples_leaf> ::= RANDINT(1,50)
#<bootstrap> ::= True | False
```

Appendix B

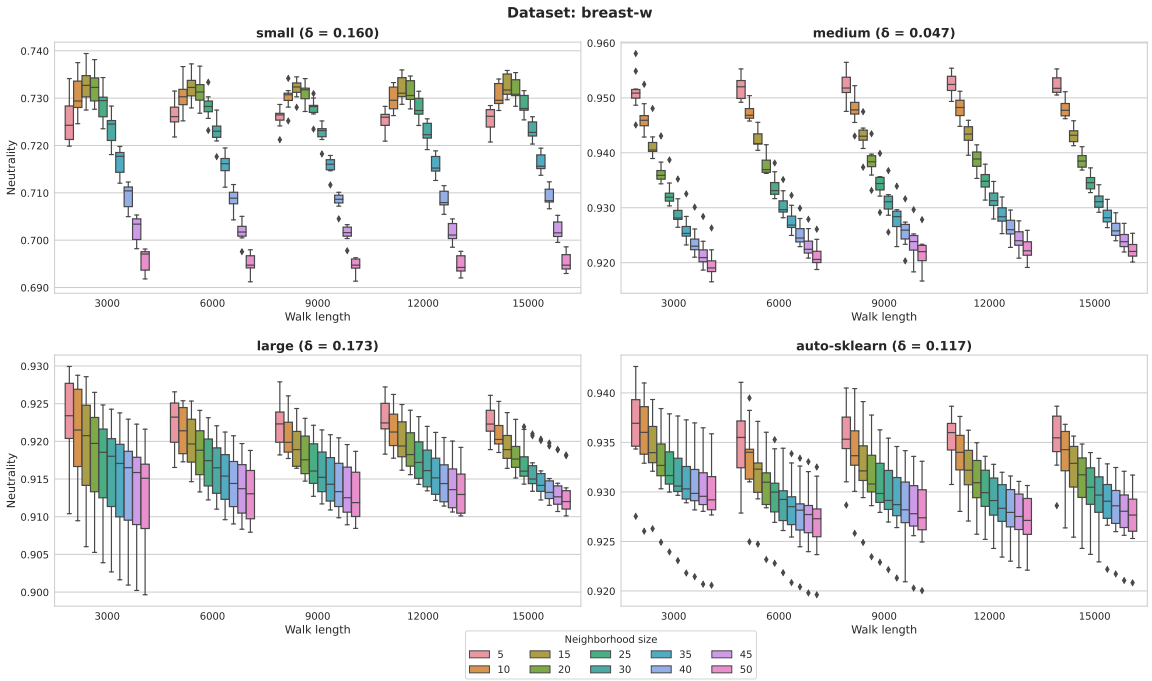
Additional Results

This appendix contains results for datasets omitted from Chapter 4.

B.1 Neutrality

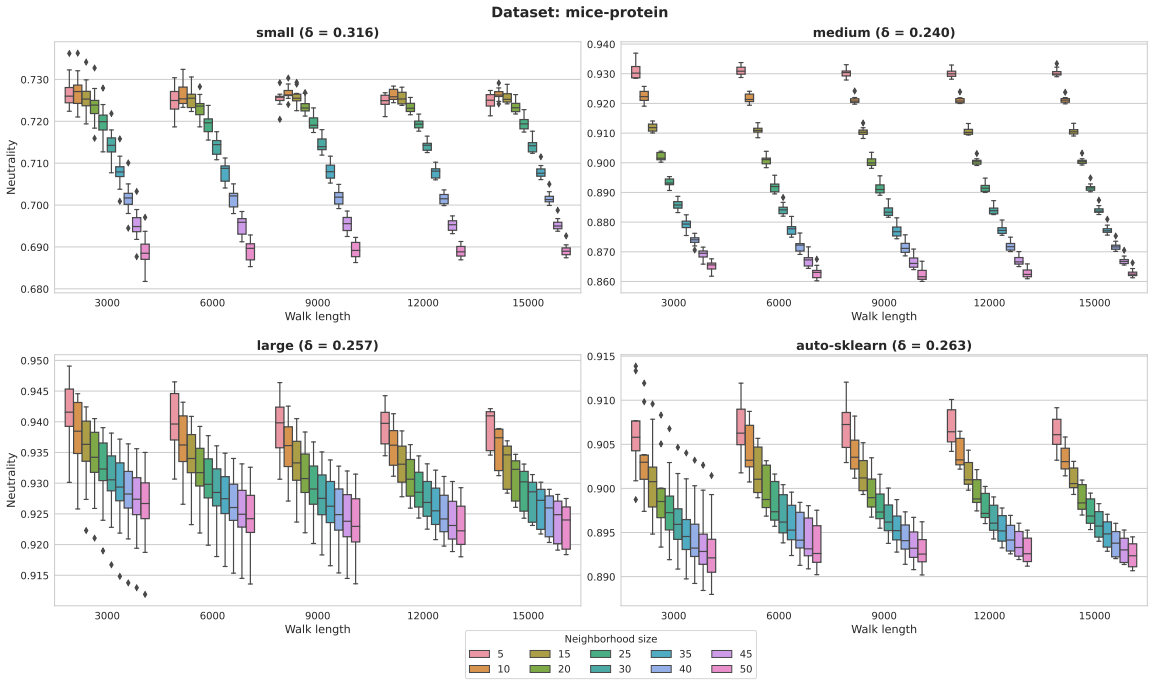
Figures B.1 to B.8 show additional neutrality results.

Figure B.1: Neutrality evaluation for the breast-w dataset.



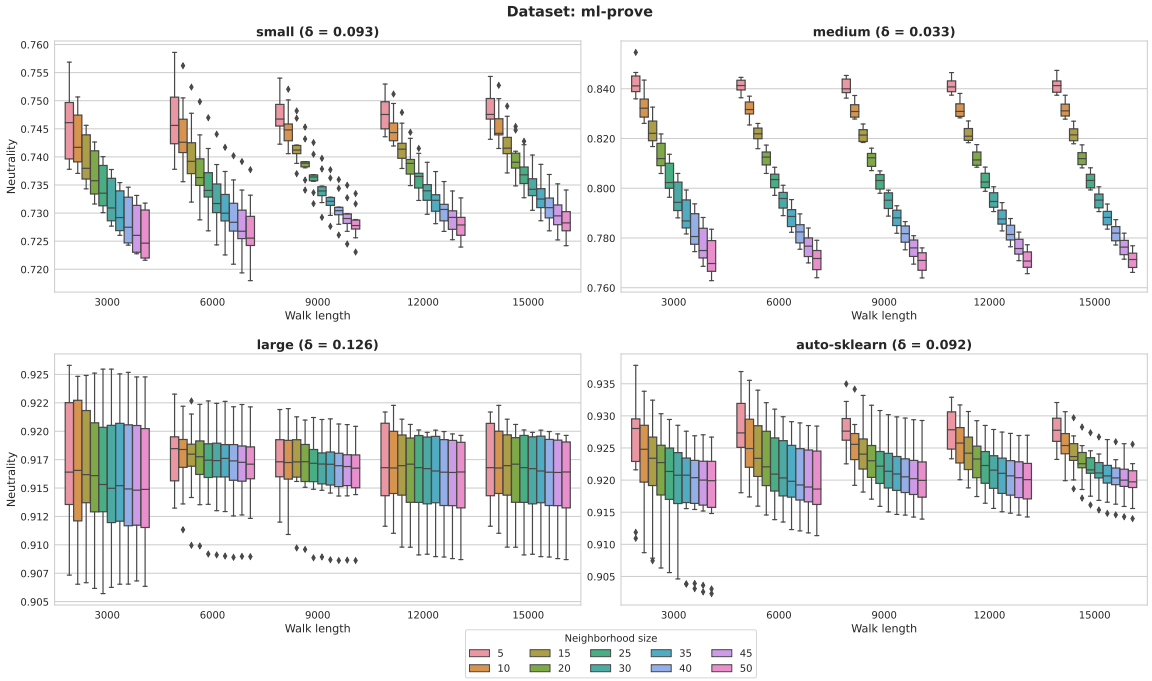
Source: author.

Figure B.2: Neutrality evaluation for the mice-protein dataset.



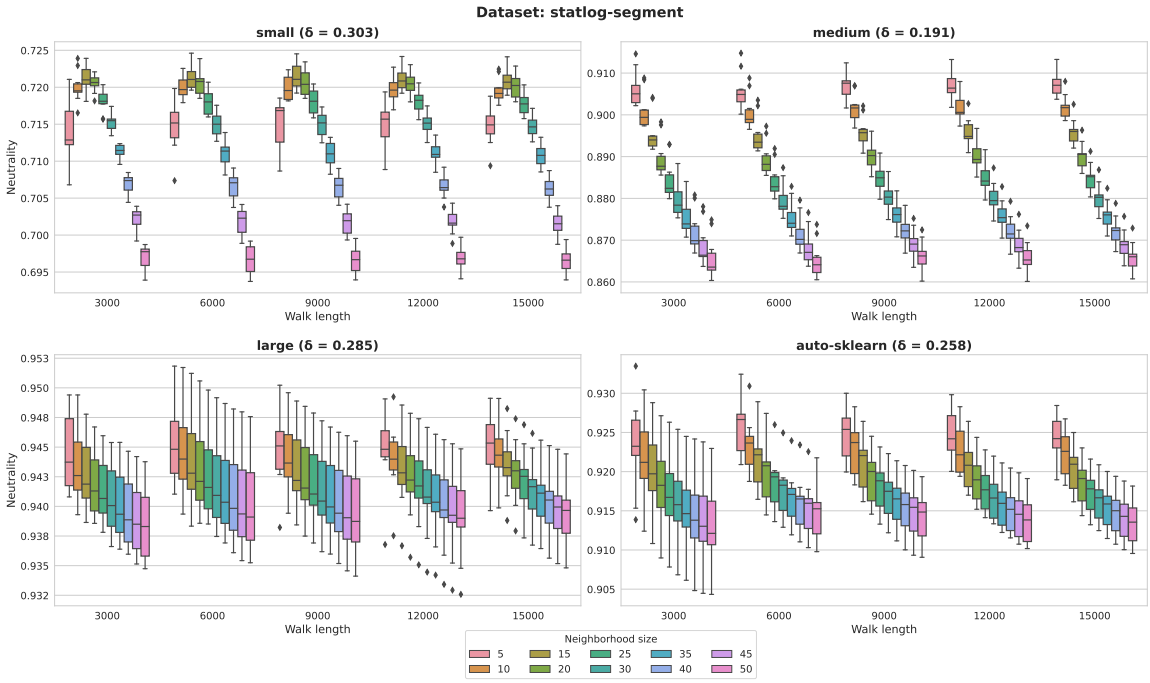
Source: author.

Figure B.3: Neutrality evaluation for the ml-prove dataset.



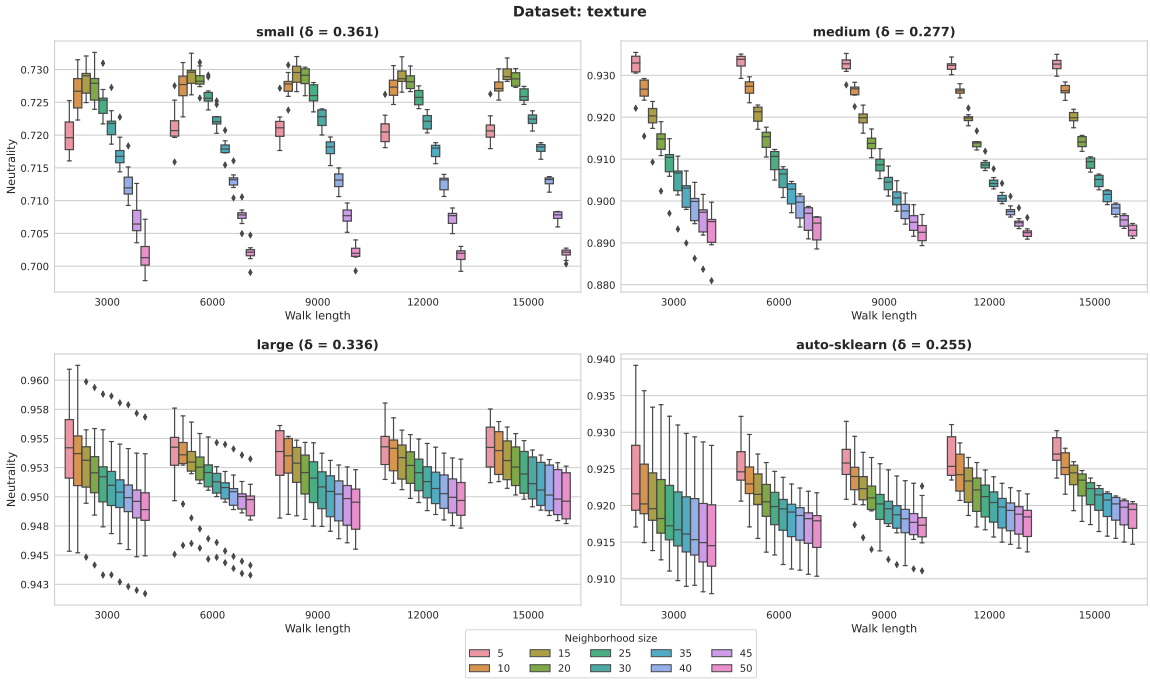
Source: author.

Figure B.4: Neutrality evaluation for the statlog-segment dataset.



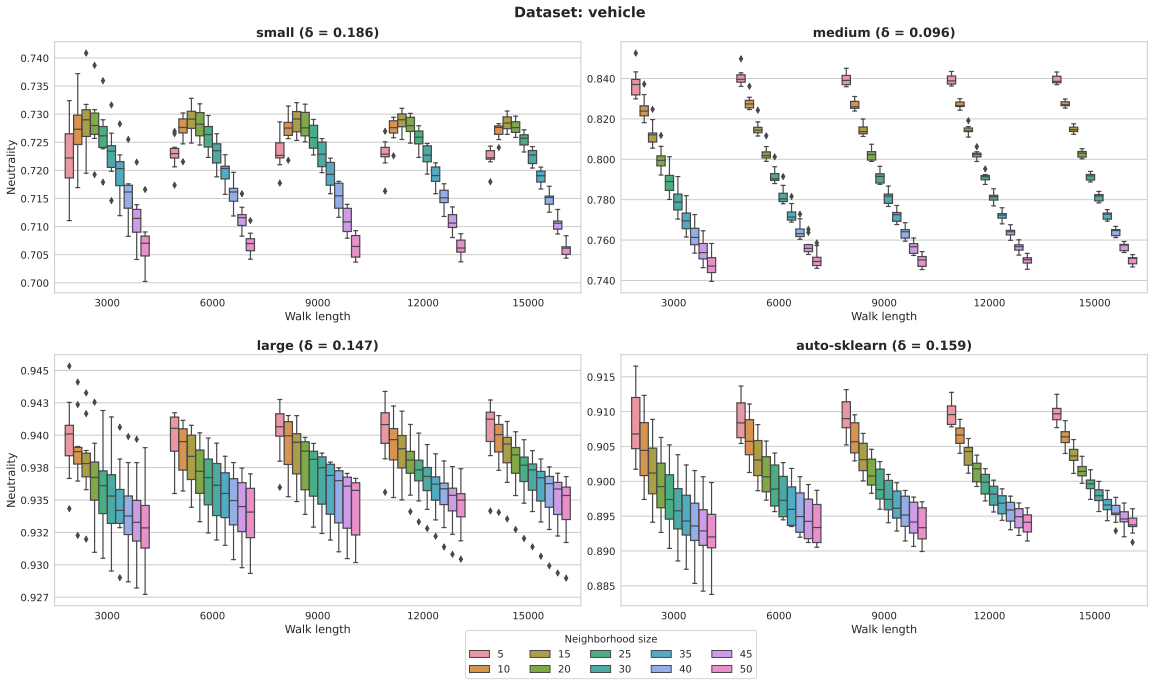
Source: author.

Figure B.5: Neutrality evaluation for the texture dataset.



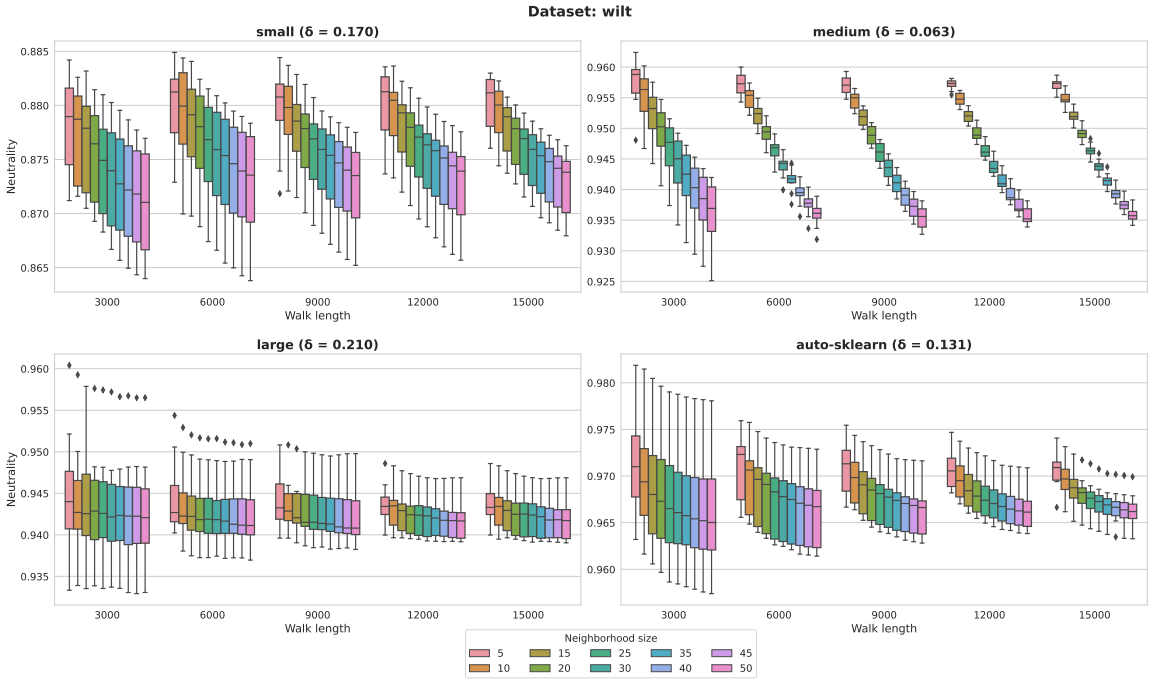
Source: author.

Figure B.6: Neutrality evaluation for the vehicle dataset.



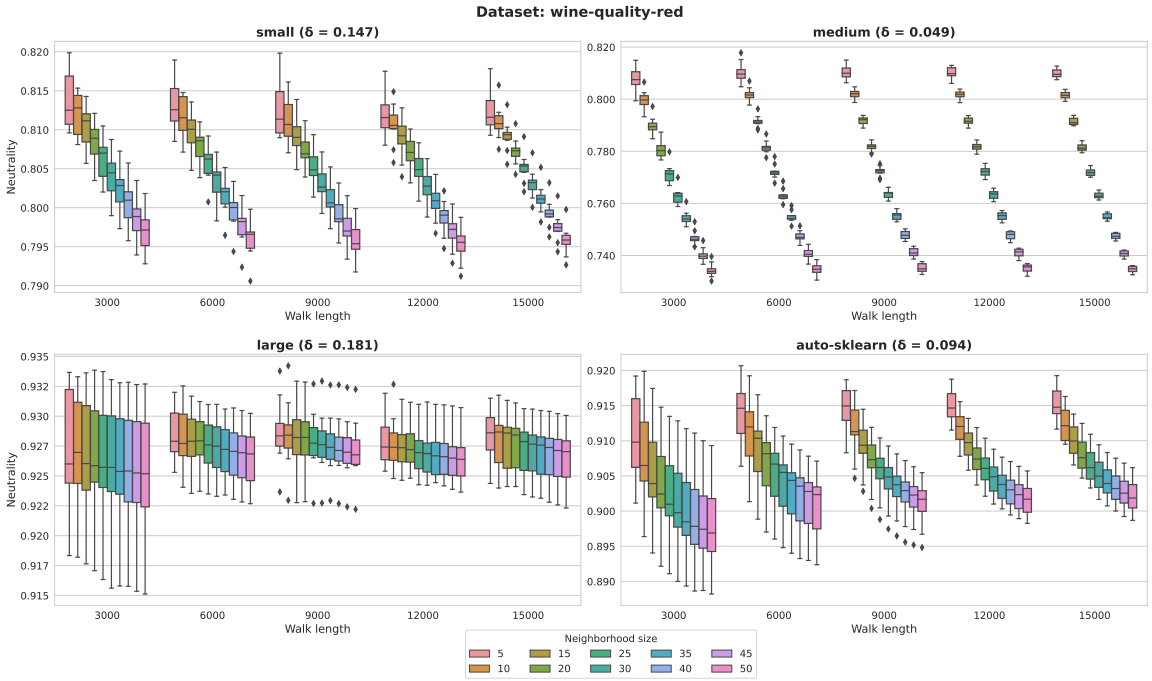
Source: author.

Figure B.7: Neutrality evaluation for the wilt dataset.



Source: author.

Figure B.8: Neutrality evaluation for the wine-quality-red dataset.

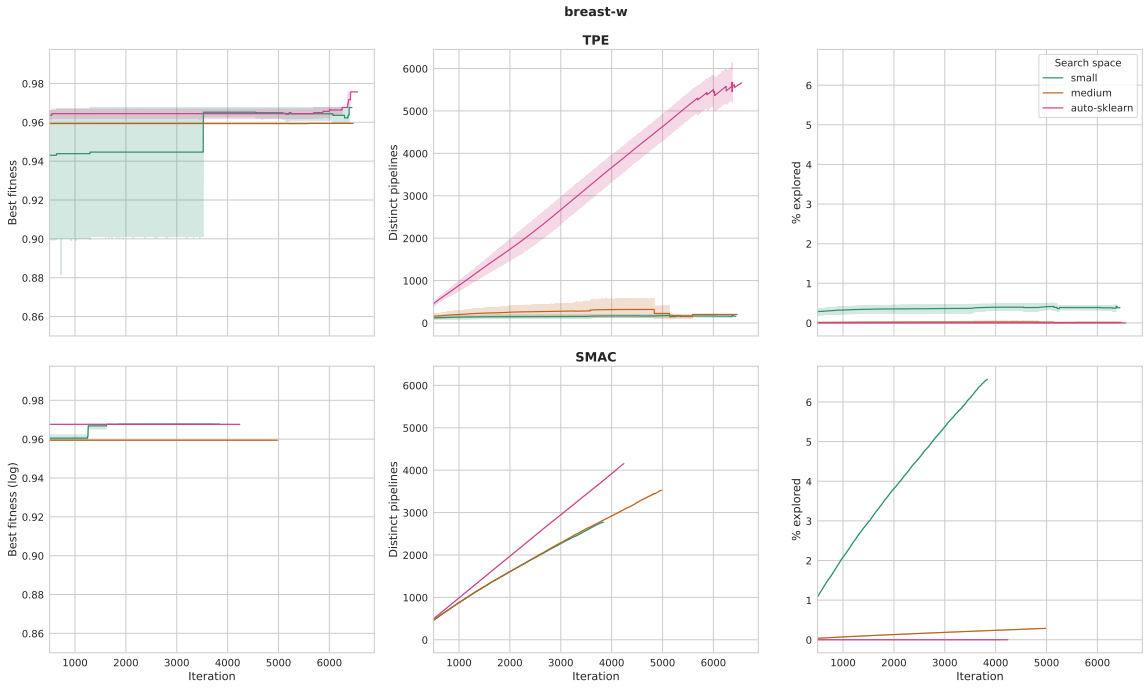


Source: author.

B.2 Search Space Exploration by AutoML Optimizers

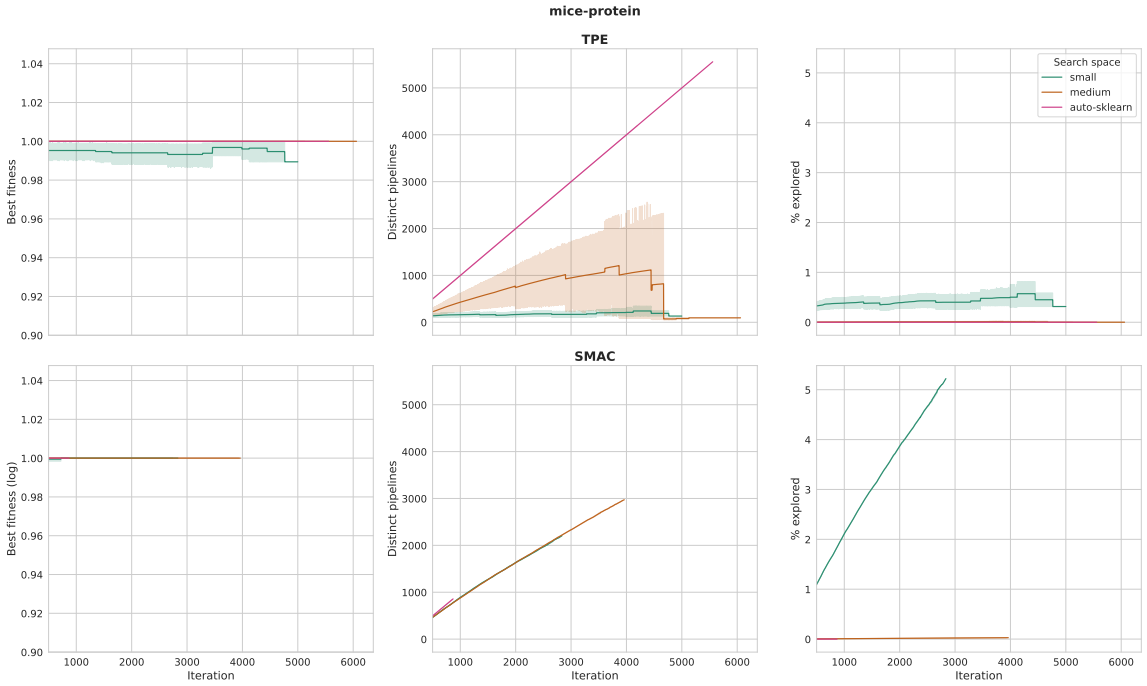
Figures B.9 to B.16 illustrate the exploration of AutoML optimizers for datasets not included in Section 4.4.

Figure B.9: Exploration of the search spaces by TPE and SMAC for dataset breast-w.



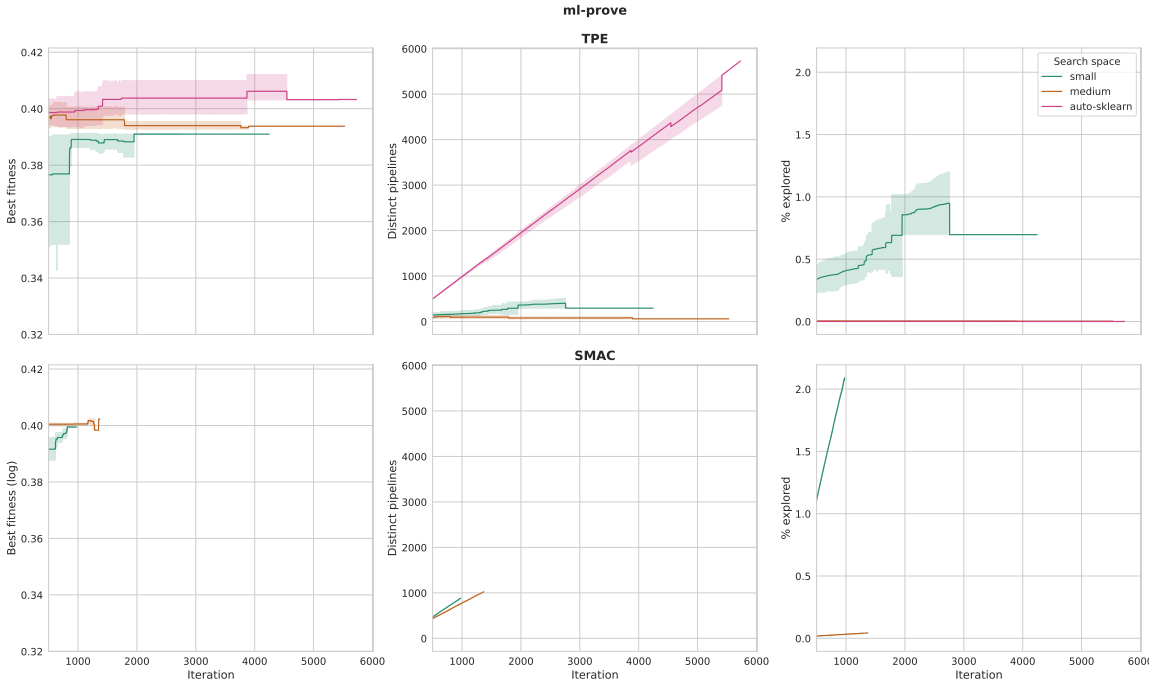
Source: author.

Figure B.10: Exploration of the search spaces by TPE and SMAC for dataset mice-protein.



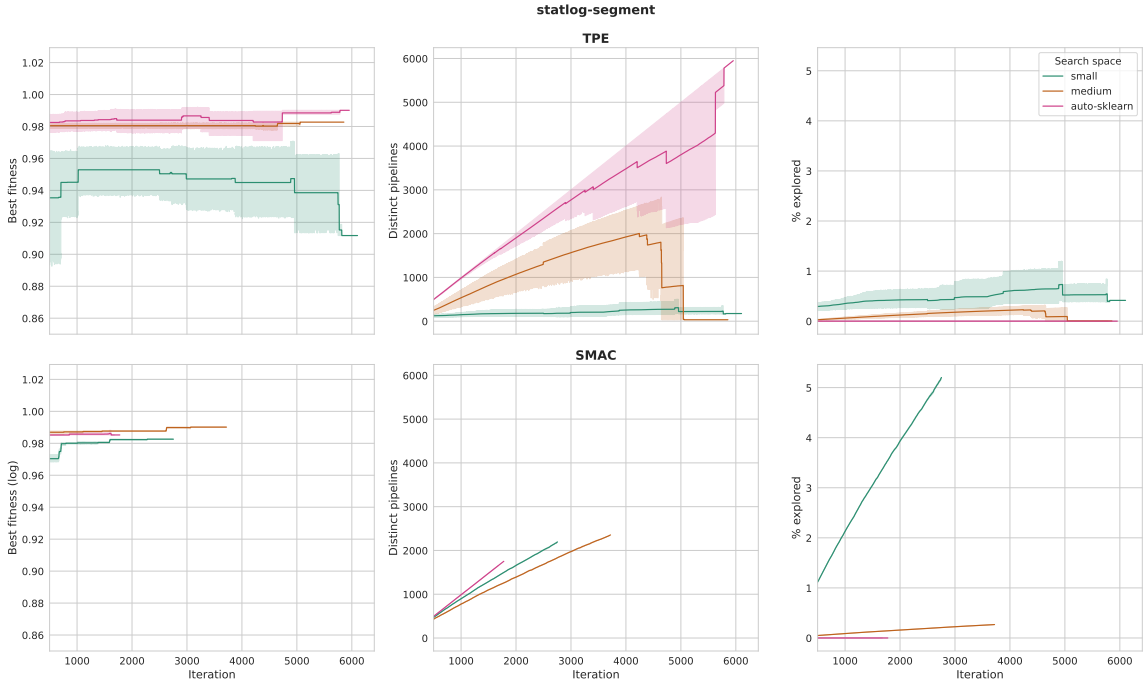
Source: author.

Figure B.11: Exploration of the search spaces by TPE and SMAC for dataset ml-prove.



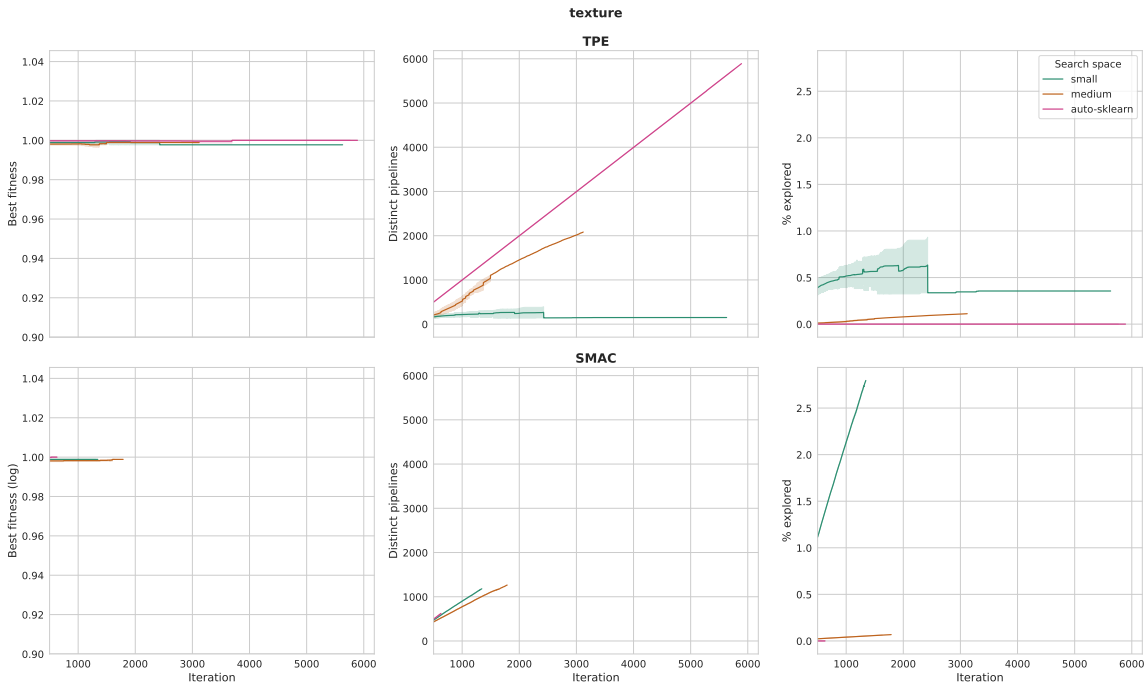
Source: author.

Figure B.12: Exploration of the search spaces by TPE and SMAC for dataset statlog-segment.



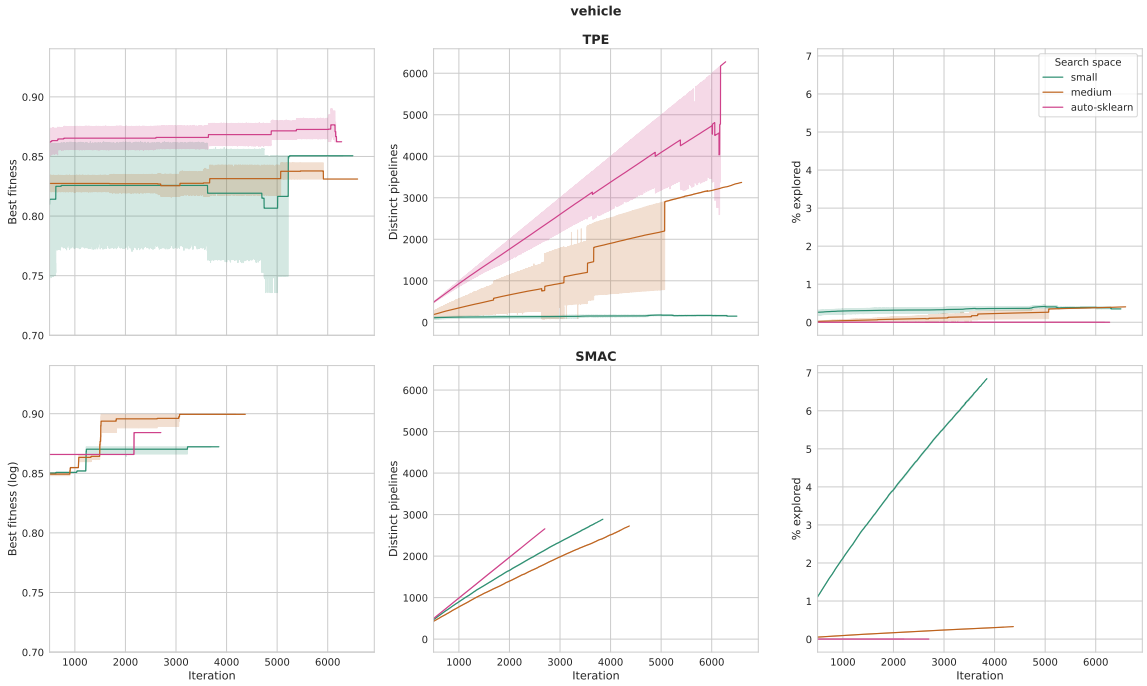
Source: author.

Figure B.13: Exploration of the search spaces by TPE and SMAC for dataset texture.



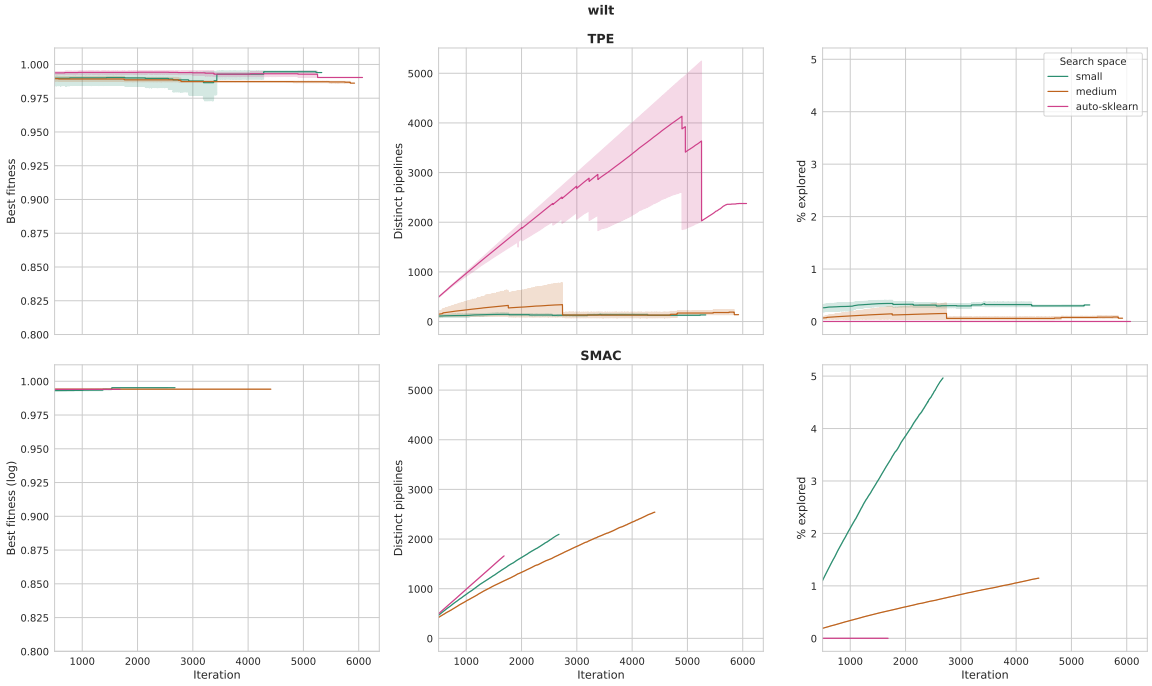
Source: author.

Figure B.14: Exploration of the search spaces by TPE and SMAC for dataset vehicle.



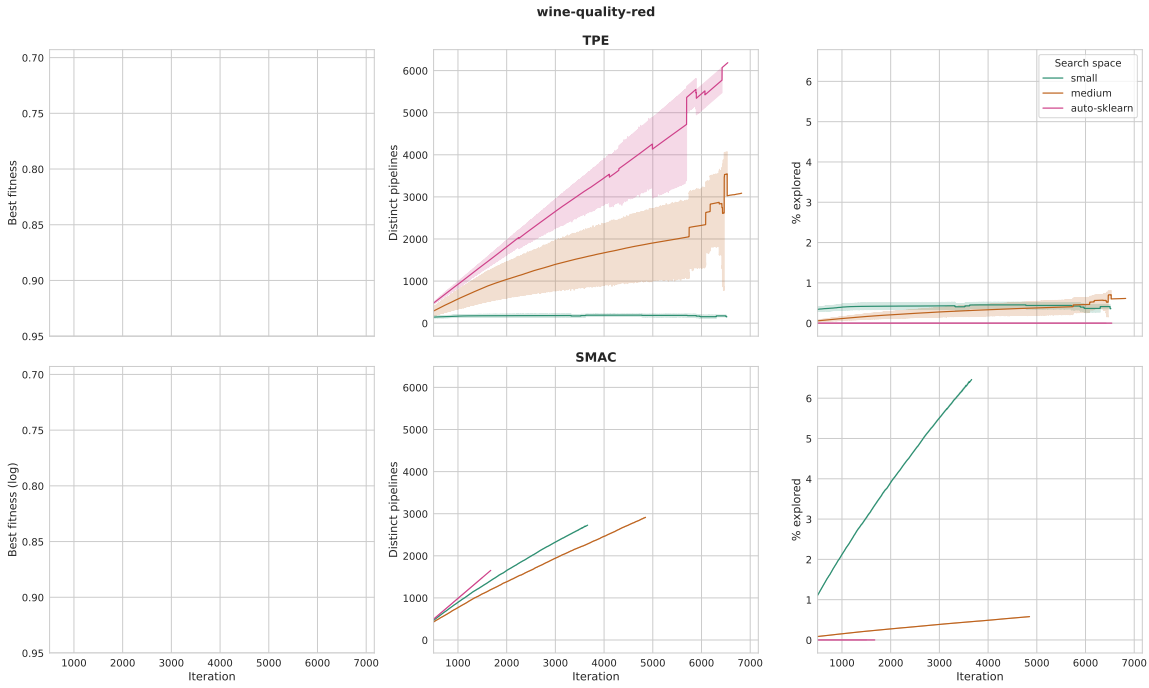
Source: author.

Figure B.15: Exploration of the search spaces by TPE and SMAC for dataset wilt.



Source: author.

Figure B.16: Exploration of the search spaces by TPE and SMAC for dataset wine-quality-red.



Source: author.