Efficient tuning of automated machine learning pipelines

Nguyen, D.A.

CHAPTER 1

Introduction

In recent years, Machine Learning (ML) has achieved success in many real-world applications, such as self-driving cars [1], assisting doctors in diagnosing diseases [2], playing video games [3], recognizing faces [4], translating languages [5], detecting automatic faults [6], and predictive maintenance [7]. In order to apply ML in real-world applications, practitioners must select a sequence of machine learning algorithms (e.g., data preprocessing, feature preprocessing, learning algorithm), together with their configurations well-suited to the target problem [8]. These tasks are always challenging due to the plethora of algorithms. Moreover, the No Free Lunch (NFL) theorem [9], [10] prescribes that there is no universally best algorithm for all problems. Therefore, the development of high-performance machine-learning applications requires highly skilled ML experts, data scientists and domain experts [11], [12]. Together, these experts attempt further experiments/trials, resulting in the best-performing ML pipeline for the given problem. In other words, applying ML to a real-world application is challenging, as it requires considerable human effort and has a strong dependency on data scientists.

The class of *Automated machine learning* (AutoML) approaches [8] are widely applied to automatically produce the best machine learning pipeline in order to minimize reliance on data scientists in the machine learning development cycle for real-world applications [8]. The domain experts can profit from using AutoML by automatically choosing a well-performing sequence of ML algorithms and their optimized hyperparameters, leading to a sensible ML model for their real-work problems without relying on ML experts.

Initially, AutoML was typically referred to as an optimization process for finding the optimal ML pipeline (with its tuned hyperparameters) for ML problems [13]. Hence, AutoML is known as a combination of Machine Learning and optimization for applying machine learning to real-world problems, that is, making ML easier to use for people without expert knowledge in ML. AutoML research mainly

focuses on optimization, whereas the ML part is inherited from the well-developed ML community. From the perspective of the optimization community, the ML part is treated as an objective configured by various ML algorithms with their hyperparameters and evaluated by an objective function for resulting in a real-valued performance, e.g., prediction accuracy or running time. A search space defines the possible choices among the algorithms and hyperparameters. Different hyperparameter settings led to different results [14]. Then, a practical optimization approach is used to find the best setting that maximizes the performance of the objective function. All the above steps are the core functions of a typical AutoML framework.

On the other hand, AutoML products are also known as a particular type of software in a ready-to-use state where the effectiveness in applying machine learning to real-world problems is the top goal [11]. Thus, modern AutoML systems incorporate many supportive functions, such as supporting the core part (e.g., applying meta-learning to find the better candidate earlier) and supporting humans to increase trust (e.g., explainable ML, visualization), and shortening the final product development cycle in production (e.g., integrating the low/no code techniques).

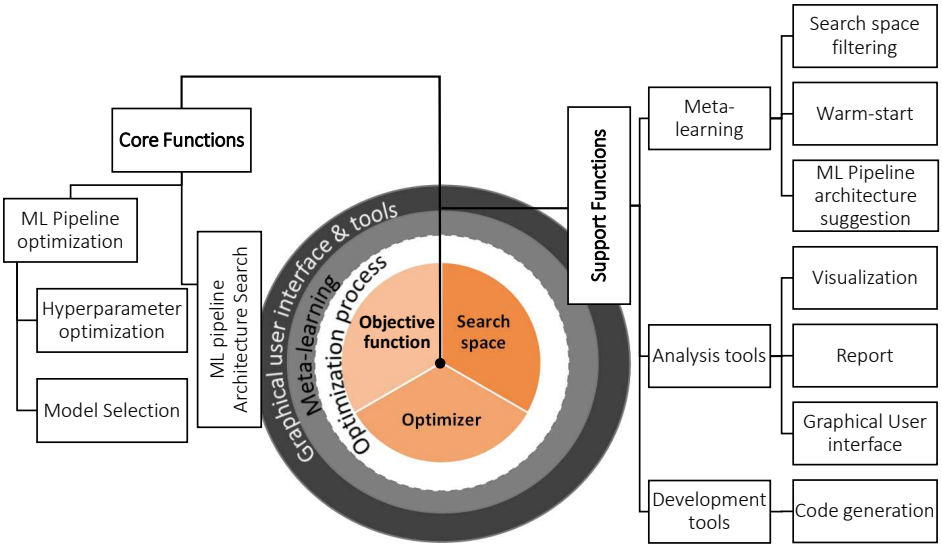

Figure 1.1: AutoML functions

Some of the core functions and support functions of a typical AutoML framework are summarized in Figure 1.1. All are discussed in this thesis, where the machine

learning pipeline optimization approaches are discussed in Chapter 3, and the remaining functions are discussed in Chapter 2, i.e., Meta-learning is discussed in Section 2.3, code generation and analysis tools for AutoML are presented in Section 2.4 and Section 2.2 presents ML pipeline architecture search. However, this thesis emphasizes the optimization of AutoML, called *AutoML optimization*, which is detailed in Section 1.1. Prior to the detailed discussions, we shall give a brief explanation of some important aspects of the AutoML optimization problem (the orange cycle in the middle of Figure 1.1).

Search space or domain space refers to the set of all possible combinations of algorithms and the algorithm's hyperparameters that the optimization algorithm can choose from. There are four common classes of hyperparameters:

1. *Ordinal hyperparameter*[1] belong to a subset of $\mathbb{Z}$, e.g., [1, 10];
2. *Continuous hyperparameter*[1] is a subset of $\mathbb{R}$, e.g., [0, 1];
3. *Nominal hyperparameter* is a set of categorical values, e.g., [Linear, RBF, Poly, Sigmoid];
4. *Hierarchical hyperparameter* or *conditional hyperparameter* is a particular type of hyperparameter, that is used to define the dependencies between hyperparameters, i.e., a hyperparameter depends on another hyperparameter;
5. In practice, the choice of algorithm is a particular type of nominal hyperparameter, which is referred to as the *algorithm choice* [15] and *one-of* nominal hyperparameter [16]. The dependent hyperparameters of an algorithm are mapped to an algorithm by a conditional hyperparameter. In this thesis, each set of algorithms for an operation function shall be called an "*operator*", e.g., a resampling operator– ([SMOTE [17], SMOTETomek [18], SVMSMOTE [19]]), learning operator–([k-nearest neighbors [20], Support Vector Machines [21]]).

Without loss of generality, let a machine learning pipeline structure be modeled as a directed acyclic graph (DAG) which is restricted by implicit constraints [22], [23], i.e., some operators are optional, which might have a *do not use* option, but the learning operator is mandatory and as the last operator. Note that we shall use the term *machine learning pipeline structure* and *sequence of algorithms over operators* interchangeably in this thesis. The complete machine learning pipeline includes a *sequence of algorithms over operators* and their dependent hyperparameters.

[1] We note that, ordinal and continuous hyperparameters are typically bounded for practical reasons.

In conclusion, the search space defines the range of hyperparameters that the optimizer can search among to find the best set of hyperparameters (i.e., the best ML pipeline).

Objective function in AutoML optimization problem is one or more performance metrics (e.g., recall, precision rate, accuracy rate), a.k.a., evaluation metrics. These metrics are used to measure and evaluate the performance of a ML pipeline (e.g., a sequence of data pre-processing, feature engineering, and a learning algorithm) on a particular task. The optimization problem is called a *single-objective problem* if it uses one performance metric. Otherwise, it is called a *multi-objective optimization problem*. Throughout this thesis, the objective function is limited to the single-objective optimization problem. In other words, an objective function is a real-valued function that measures the performance of the ML pipeline on a given dataset using a specific performance metric. The objective function can be constructed using cross-validation, ensemble methods, or other techniques.

Optimization algorithm applied to AutoML is an algorithm that aims to find the optimal machine learning pipeline on a given dataset using a performance metric, i.e., objective function. Optimizing the objective function involves a trade-off between computational complexity and accuracy. The choice of algorithm depends on the problem's complexity, the dataset's size, and the computational resources available. The literature has highlighted several possible optimization approaches that can be used for AutoML, such as:

- *Bayesian optimization* (BO), e.g., Hyperopt [24], SMAC [25], [26], SPO [27], Spearmint [28], BO4ML [15].
- *Grid search* [29] and *Random search* (RS) [30], [31].
- *Racing procedure* (RP) [32], [33].
- *Bandit learning* (BL), e.g., Successive Halving [34], HyperBand [35].
- *Hybrid approaches*, e.g., BOHB [36] and DACOpt [37] hybridize Bayesian optimization and bandit learning, MIP-EGO [38] hybridizes Bayesian optimization and evolutionary algorithms.

However, Bayesian Optimization is the most commonly used approach in this domain as it was used in Auto-sklearn [39], Auto-Weka [40], [41] and Hyperopt-sklearn [42]. Hence, this thesis will review all mentioned approaches but mainly focus on improving BO and using it to solve the AutoML optimization problems.

1.1 Problem definition

In this thesis, the discussion is focused on AutoML optimization (AO) problems, and the ML problem is limited to supervised machine learning fields, i.e., regression and classification problems. Given a classification problem with a dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, where $\mathbf{x} = \{x_1, \ldots, x_k\}$ is a vector representation of the k features x, and y represents a label in classification problem or a continuous value in a regression problem. Solving an AutoML optimization problem refers to finding the best ML model $\mathrm{P} : \mathbb{X} \to \mathbb{Y}$ that transforms a vector $\mathbf{x} \in \mathbb{X}$ into a target value $y \in \mathbb{Y}$. In AutoML optimization, the ML model refers to as a ML pipeline model, which can include data pre-processing, encoding, feature selection, resampling, and learning algorithm. Hence, we shall interchange the terms of *ML model* and *ML pipeline model* in this section.

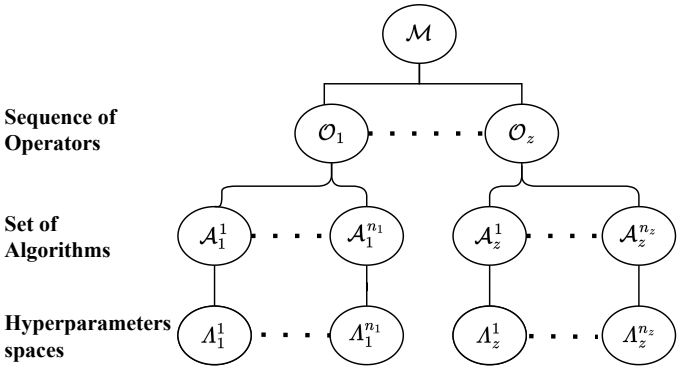

Figure 1.2: The structure of AutoML search space

Let $\hat{y}$ denote the predicted label calculated by model P for input features $\mathbf{x}$, i.e., $\hat{y} = \mathrm{P}(\mathbf{x})$, and $R(\hat{y}, y)$ denote a measure of classification accuracy between the predicted label $\hat{y}$ and the true label y. Let p be an ML pipeline setting trained on dataset $\mathcal{D}$ to produce the ML model P, i.e., $\mathrm{P} = (p, \mathcal{D})$. The performance of p is then calculated as:

$$\begin{aligned} f(p, \mathcal{D}) &= \frac{1}{m} \sum_{j=1}^{m} R(\mathrm{P}(\mathbf{x}_i), y_j) \\ &= \frac{1}{m} \sum_{j=1}^{m} R(\hat{y}_j, y_j) \end{aligned} \tag{1.1}$$

In practice, the predictive performance of ML pipeline p needs to be calculated based on its prediction on an unseen dataset, i.e., a test/validation set. The dataset $\mathcal{D}$ is then split into non-overlapping training and validation sets: $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ and $\mathcal{D}_{\text{valid}} = \{(\mathbf{x}_{n+1}, y_{n+1}), \ldots, (\mathbf{x}_m, y_m)\}$. Equation 1.1 allows calculating performance of the ML pipeline setting p when trained on $\mathcal{D}_{\text{train}}$ and evaluated on $\mathcal{D}_{\text{valid}}$ as:

$$f(p, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}}) = \frac{1}{m-n} \sum_{j=1}^{m-n} R(\hat{y}_{n+j}, y_{n+j}) \tag{1.2}$$

However, the over-fitting problem is often encountered in applying optimization approaches [13]. To overcome this problem, the k-fold cross-validation can be applied to the Equation. 1.2, which becomes:

$$f(p, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}}) = \frac{1}{k} \sum_{j=1}^{k} f(p, \mathcal{D}^j_{\text{train}}, \mathcal{D}^j_{\text{valid}}) \tag{1.3}$$

where $f(p, \mathcal{D}^j_{\text{train}}, \mathcal{D}^j_{\text{valid}})$ is performance of the ML pipeline p when trained and evaluated on the j^{th} data fold $\mathcal{D}^j_{\text{train}}$ and $\mathcal{D}^j_{\text{valid}}$, correspondingly.

Solving an AutoML optimization problem involves finding the optimal ML pipeline p that trains on a dataset $\mathcal{D}$ to produce the best-performing ML model P. Traditionally, the AutoML optimization problem is customarily treated and solved as a hyperparameter optimization problem (HPO), called the *HPO-based approach*, where the target ML pipeline p is constructed as a single algorithm. An alternate approach is to construct the target ML pipeline p structure as an ML pipeline of multiple ML algorithms, called *ML pipeline-based approach*. In this section, we will first look at solving the problem of AutoML optimization, also referred to as ML pipeline optimization. This topic is explained in detail in Section 1.1.1. Afterward, we will briefly discuss the traditional HPO-based approach, which treats the problem as a hyperparameter optimization problem. This discussion is covered in Section 1.1.2.

1.1.1 Machine Learning Pipeline Optimization

Let $\mathbb{O} = \mathcal{O}_1 \times \ldots \times \mathcal{O}_z$ denote the sequence of operators in the pipeline p, where each subsequent operator is applied to the output of the previous operator starting from input $\mathbf{x}$: $p(\mathbf{x}) = \mathcal{O}_z(\mathcal{O}_{z-1}(\ldots(\mathcal{O}_1(\mathbf{x}))\ldots))$. The functionality of each such operator can typically be delivered by one of the multiple available ML algorithms: herein, we assume $\mathcal{O}_{i\in\{1,\ldots,z-1\}} = \{\varnothing, \mathcal{A}^1_i, \ldots, \mathcal{A}^{n_i}_i\}$ for all operators except the

last and $\mathcal{O}_z = \{\mathcal{A}_z^1, \ldots, \mathcal{A}_z^{n_z}\}$ for the last operator which defines the learning algorithm – i.e., unlike the first $z-1$ operators, the last operator $\mathcal{O}_z$ has to be selected and cannot be $\varnothing$.

Typically, all algorithms have hyperparameters; we denote the domain of the i-th hyperparameter by $\boldsymbol{\lambda}_i$. Let $\Lambda^i = \boldsymbol{\lambda}_1 \times \ldots \times \boldsymbol{\lambda}_b$ be a hyperparameter space consisting of b hyperparameters of algorithm $\mathcal{A}^i$. Let $\Lambda_{\mathcal{O}_i} = \Lambda_i^1 \cup \ldots \cup \Lambda_i^{n_i}$ be the hyperparameter space of the operator $\mathcal{O}_i$ and $\Lambda = \Lambda_{\mathcal{O}_1} \cup \ldots \cup \Lambda_{\mathcal{O}_z}$ denotes the hyperparameter space of all considered algorithms for all considered operators. Let $\mathcal{M}$ denote the search space that includes the sequence of operators $\mathbb{O}$ and its corresponding hyperparameter spaces. The overall structure of the resulting AutoML search space is illustrated in Figure 1.2.

For readability, let $\mathcal{A}_{i,\lambda_j}$ represent algorithm $\mathcal{A}$ selected for an operator $\mathcal{O}_i$ and configured by a hyperparameter setting $\lambda_j \in \Lambda_{\mathcal{O}_i}$. Then, we denote a pipeline with algorithms selected and configured with their hyperparameters for all operators in the pipeline p as $p_{(\mathcal{A}_{1,\lambda_1},\ldots,\mathcal{A}_{z,\lambda_z})}$. We note that some approaches [43] consider in finding a suitable order of algorithms in the pipeline, which is called ML pipeline structure selection [22], [43]. Let the set of valid ML pipeline structures be denoted by $S = \{s_1, ..., s_h\}$, where $s \in S$ represents a valid ML pipeline structure that orders the position of algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_z$ in the pipeline. The ML pipeline becomes $p_{(s,\mathcal{A}_{1,\lambda_1},\ldots,\mathcal{A}_{z,\lambda_z})}$. However, it is worth noting that most AutoML optimization approaches do not search for structure, such as Auto-sklearn [39], Auto-weka [40], Hyperopt-sklearn [42]. Instead, they use a fixed structure, i.e., $|S|= 1$. This fixed structure is based on a well-known linear sequence of operators recommended in literature or based on their experiences. When $|S|> 1$, it is referred to as the case of flexible ML pipeline structure search. This thesis limits our discussion to the fixed ML pipeline structure. Therefore, we will use the notation $p_{(\mathcal{A}_{1,\lambda_1},\ldots,\mathcal{A}_{z,\lambda_z})}$ to denote the ML pipeline, while the ML pipeline structure will not be further discussed. However, we will briefly discuss both approaches in Chapter 2, i.e., the discussion on the Fixed ML pipeline structure in Section 2.2.1 and the Flexible ML pipeline structure in Section 2.2.2.

To solve the AutoML problem (see Equation 1.4) and find the best choice of algorithms and their hyperparameters for the pipeline operators, every such choice needs to be evaluated. The $R(\hat{y}, y)$ denotes a metric that returns the accuracy of value $\hat{y}$ predicted by the pipeline compared to the real value y. Then, performance f of pipeline configuration $p_{(\mathcal{A}_{1,\lambda},\ldots,\mathcal{A}_{z,\lambda})}$ when trained on a training dataset $\mathcal{D}_{\text{train}} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ and evaluated on a validation dataset $\mathcal{D}_{\text{valid}} =$

$\{(x_{m+1}, y_{m+1}), \ldots, (x_{m+t}, y_{m+t})\}$ is calculated as: $f(p_{(\mathcal{A}_{1,\lambda},\ldots,\mathcal{A}_{z,\lambda})}, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}}) = \frac{1}{t}\sum_{j=1}^{t} R(\hat{y}_{m+j}, y_{m+j})$. Hence, the AutoML optimization problem becomes the ML pipeline optimization maximizing problem:

$$(\mathcal{A}_{1,\lambda_1}, \ldots, \mathcal{A}_{z,\lambda_z})^* = \underset{\mathcal{A}_{1,\lambda_1},\ldots,\mathcal{A}_{z,\lambda_z}}{\operatorname{argmax}} f\Big(p_{(\mathcal{A}_{1,\lambda_1},\ldots,\mathcal{A}_{z,\lambda_z})}, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}}\Big) \tag{1.4}$$

where $(\mathcal{A}_1, \ldots, \mathcal{A}_z) \in \bigtimes_{i=1}^{z} \mathcal{O}_i$ are all possible choices of algorithms for all pipeline operators, $\{\lambda_1, \ldots, \lambda_z | \lambda_1 \in \Lambda_{\mathcal{O}_1}, \ldots, \lambda_z \in \Lambda_{\mathcal{O}_z}\}$ are algorithms' hyperparameters and $f\Big(p_{(\mathcal{A}_{1,\lambda_1},\ldots,\mathcal{A}_{z,\lambda_z})}, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}}\Big)$ is performance of the sequence operators and their corresponding hyperparameter choices when trained and evaluated on $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{valid}}$[2], correspondingly.

1.1.2 Combined Algorithm Selection and Hyperparameter Optimization

Traditionally, the AutoML optimization problem is commonly referred to as Combined Algorithm Selection and Hyperparameter Optimization (CASH) [39], [40] or Full Model Selection (FMS) [44] problem, in which the choice of algorithm is modeled as an additional categorical hyperparameter. Then, the AutoML optimization problem is treated as a HPO problem. As such, the choice of algorithms for each operator is modeled as an extra categorical hyperparameter $\boldsymbol{\lambda}^0$. The search space for the i^{th} operator is then defined as $\Lambda_{\mathbb{O}_i} = \boldsymbol{\lambda}_i^0 \cup \Lambda_i^1 \cup \ldots \cup \Lambda_i^{n_i}$, and the entire search space be $\Lambda = \Lambda_{\mathbb{O}_1} \cup \ldots \cup \Lambda_{\mathbb{O}_z}$ Hence, the AO problem becomes the HPO maximizing problem:

$$\lambda^* = \arg\max_{\lambda \in \Lambda} f(\lambda, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}}) \,, \tag{1.5}$$

where $f(\lambda, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}})$ is performance of the hyperparameter setting $\lambda \in \Lambda$ when trained and evaluated on $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{valid}}$, correspondingly.

In this setting, the categorical hyperparameters after the root of this hierarchical search space (see Figure 1.2) are known as the choice of algorithm for an operator. Consequently, algorithms and their local hyperparameters are treated at the same

[2]We note that AutoML tools are typically evaluated based on their performance on an unseen dataset during optimization, e.g., test set or ground truth set. These tools include several strategies to avoid overfitting together with other setups. These comparisons compare the performance of the whole AutoML system rather than the optimizer only. However, this thesis focuses on the optimization process to maximize performance on an unseen dataset for ML algorithms but known to the optimizer, referred to here as the validation set $\mathcal{D}_{\text{valid}}$. Therefore, we will not use the term test set $\mathcal{D}_{\text{test}}$ in this section. Instead, $\mathcal{D}_{\text{test}}$ will be used in Chapter 4 (Section 4.3), where we set up benchmarks for comparing different AutoML tools.

level. However, unlike the pure categorical hyperparameter, i.e., choose one in a set of nominal options, the choice of algorithms heavily affects other hyperparameters, i.e., once the algorithm is known, only its hyperparameters are relevant.

Another point worth mentioning is that HPO was originally developed to find the best hyperparameter setting from a single algorithm which is a much more straightforward problem compared to the AutoML optimization problem. In addition to HPO, AutoML optimization also searches for an optimized pipeline of algorithms. In AutoML, multiple algorithms must be considered, and these algorithms can belong to different phases in the ML pipeline, for example, pre-processing and learning models. This pipeline is restricted by some constraints, such as the learning task, i.e., classification and regression for supervised learning and clustering for unsupervised learning, which is the last step. For example,

- Auto-sklearn [39], [45] has up to six sequence operation steps: categorical encoder, numerical transformer, imputation transformer, rescaling, feature preprocessor, and learning operator.

- In comparison, Auto-Weka [40], [41] and Hyperopt-sklearn [42] have only two operators: preprocessor and learning operator.

Although, in general, AutoML can have different sizes in terms of operators and algorithms under operators, most operators are optional, and the learning operator is mandatory.

Furthermore, the algorithms and techniques used in an ML pipeline are tightly coupled because every operator step is directly affected by the previous step. For example, the data pre-processing step aims to produce a new dataset (balanced, reduced-dimensions, etc.), which can change the performance of the subsequent operator, such as the learning model. Consequently, the traditional approach for handling the choice of algorithm is a mismatch with the nature of the AutoML optimization problem.

1.2 Research Questions

In this thesis, we focus on AutoML techniques that aim at shorting the progress of producing ML applications. Some of the most critical questions that we will try to address are:

RQ1: How can we automatically construct high-quality ML pipelines for imbalanced data with HPO algorithms?

The classification algorithms commonly assume that the input data is equally distributed between classes. However, the distribution of classes in many real-world classification problems is ordinarily unequal, which reduces classification performance. To address this problem, we can apply a well-suited resampling technique to balance the imbalanced data before passing it to the classifier for training. There is no universal best algorithm for all problems. Hence, it becomes the model selection problem of finding the optimal combination of a resampler and classifier for the given problem, each selected from the sets of existing resampling techniques and classification algorithms. Besides, both resampling techniques and classification algorithms have local hyperparameters that need to be tuned to achieve better performance. Therefore, the *Model selection* (MS) and *Hyperparameter optimization* (HPO) tasks have to be considered. We note that HPO algorithms are initially designed for tuning hyperparameters of a single ML algorithm. The *Combined Algorithm Selection and Hyperparameter Optimization* (CASH) is a well-known approach for solving those two tasks simultaneously. CASH converts MS and HPO into a single HPO problem, which HPO algorithms can solve. Nevertheless, CASH has yet to be studied in detail for imbalanced class problems. Hence, we explore the potential of applying HPO algorithms to construct well-performing ML pipelines for imbalanced data automatically. Study on that problem gives us insights into how to design CASH experiments for class-imbalanced problems, such as determining applicable resampling and classification algorithms, constructing search space, and selecting performance metrics. The exploration and resolution of this research question are comprehensively tackled in Chapter 4 of this thesis.

RQ2: What is the most effective CASH optimization approach to achieve the optimal ML pipeline model for imbalance classification problems?

The Combined Algorithm Selection and Hyperparameter (CASH) problem can be addressed by a HPO algorithm. The two well-known HPO algorithms– Bayesian optimization and random search - are considered in our investigation to provide insights into choosing appropriate techniques for solving this CASH

for class imbalance problems. The comprehensive exploration and resolution of the research question can be found in Chapter 5 of this thesis.

RQ3: How to apply the CASH optimization to the steel surface defects classification problem where the distribution between classes is imbalanced and has unequal importance?

In collaboration with Tata Steel Europe - The Netherlands in the steel surface defects detection problem, we address the classification problem where the distribution between classes is imbalanced and has unequal importance, i.e., detecting severe defects that might heavily affect the quality of the final product is more priority than others with lighter affections. In CASH optimization problems, the proper performance metric is vital for evaluating ML models to correctly choose the optimal ML model. For the imbalance problem, several performance metrics (e.g., F1, geometric mean) can be used. However, those performance metrics treat all classes equally important. Thus, those metrics' overall accuracy cannot be used for this situation. In other words, the required performance metric for this real-world problem has yet to be. Therefore, to apply CASH to the steel surface defects detection problem, we need a new performance metric that considers both class imbalanced and unequal class importance problems. In Chapter 6, we delve into the heart of the research question, presenting a detailed analysis and solution.

RQ4: How does maximizing coverage of initial sampling improve BO performance to AutoML optimization problems? Bayesian optimization (BO) is a typical optimization approach that is structured by three fundamental components: initial sampling, surrogate model and acquisition function. The initial sampling step is typically restricted to a small budget since the effectiveness of BO becomes evident mainly in the later stages of optimization when it learns to produce better ML pipeline configurations. Another point worth mentioning is that the search space of AutoML is large, which includes many possible algorithms in the ML pipeline.

Many studies [46]–[48] noted that some algorithms have similar technical behaviours. To take advantage of this, we explore the potential of sampling on the group of similar algorithms for maximizing coverage of the AutoML search space already within a small budget of the initial sampling of BO. The exploration also provides insights into the effectiveness of optimized initial sampling to BO to characterize the response surface more accurately

and how we can adapt BO to solve the AutoML problems. For a detailed response to the research question, readers are directed to Chapter 7, where an exhaustive exploration and conclusion await.

RQ5: When should we stop tuning in an area of the search space?

As mentioned in **RQ4**, the AutoML search space is ample; trying every configuration is costly and typically impossible in practice due to the limited computation resources. The solution to **RQ4** can be adapted to separate the AutoML search space into multiple search areas (i.e., sub-spaces). Given limited computational resources for optimization, allocating more resources to the most promising areas while reducing resources to the unpromising areas to ensure achieving the best performance accuracy is necessary. Thus, the question is how to detect unpromising areas early and correctly. Chapter 8 provides an in-depth examination and resolution of the research question that guides this study.

RQ6: How can we efficiently allocate computational resources in the AutoML search space? The class of multi-fidelity approaches (Chapter 3. Section 3.2) aims to maximize the number of configurations to be evaluated within a limited budget. The central idea is to save computation resources for ineffective configurations and use them for other configurations. In this setting, the effective configuration will be tested on more data than the ineffective one. Racing procedures (Chapter 3. Section 3.2.1) and bandit learning (Chapter 3. Section 3.2.2) are the two well-known multi-fidelity approaches for HPO problems. Adopting their techniques to the AutoML optimization problem can provide insight into how we can handle resources efficiently in AutoML optimization. The comprehensive exploration and resolution of the research question can be found in Chapter 8 of this thesis.

1.3 Outline of the Dissertation

This thesis is organized as illustrated by a high-level overview in Figure 1.3. The motivation, research questions, and major contributions of each chapter are briefly introduced.

The relevant technical *background* for this thesis is split into two chapters where the AutoML optimization approaches are discussed in Chapter 3 and the relevant

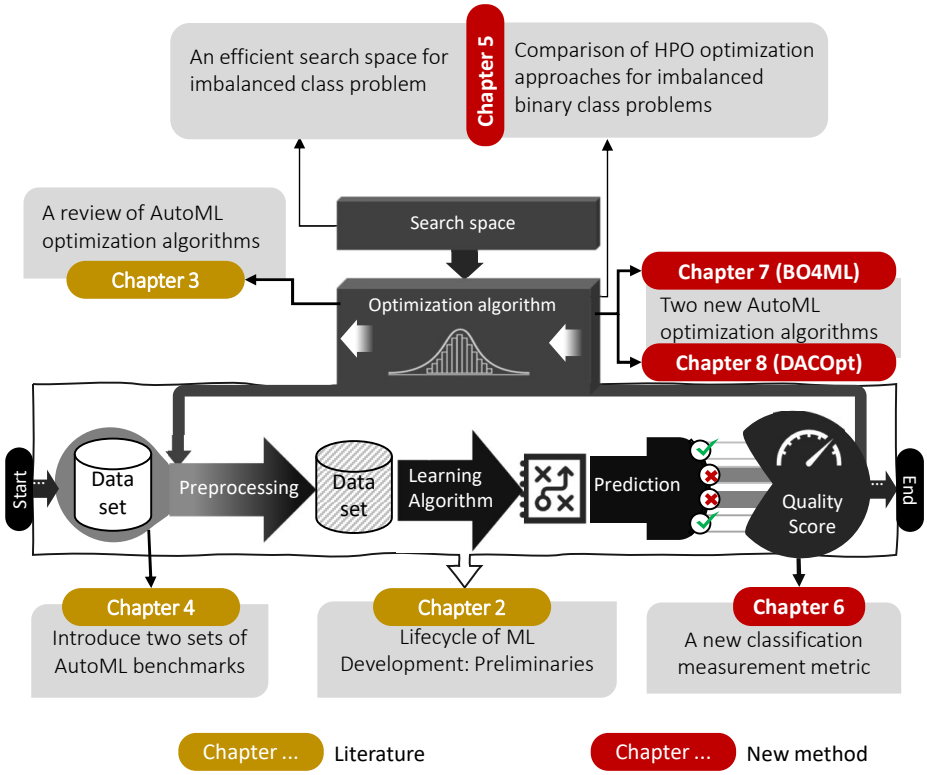

Figure 1.3: Mapping of chapters of this thesis to the process of development and life cycle of an AutoML pipeline.

machine learning development life-cycle knowledge provided by existing AutoML frameworks are discussed in Section 2.

Chapter 4 aims at introducing two sets of AutoML benchmarks to evaluate optimization approaches. Both scenarios provide a search space and a set of corresponding benchmark datasets– the first scenario includes two operators in the pipeline and 44 binary imbalanced benchmark datasets; the second scenario is a ML pipeline optimization of a dynamic search space with up to six operators and 73 AutoML benchmark datasets.

Chapter 5 focuses on the issue of how to apply optimization approaches to class imbalanced classification problems properly. Optimizer is a core component of an AutoML framework. In addition, various optimization approaches are compared in this chapter, where the most efficient approach is discovered.

Chapter 6 introduces a new classification measurement metric for the multi-class

classification problem where some classes are more important than others.

Chapter 7 and Chapter 8 discuss how to efficiently use BO for AutoML problems. Chapter 7 introduces a novel initial sampling strategy, *Combination-based sampling*, which is particularly designed for using BO for AutoML optimization problems. In addition, a novel BO approach, **BO4ML**, is proposed, where the proposed initial sampling approach is integrated. Chapter 8 introduces a novel contesting procedure, **D**ivide **A**nd **C**onquer **Opt**imization (**DACOpt**), which is an optimization approach specially designed for dealing with the large and complex search space of AutoML to help BO focus on promising search area earlier.

1.4 Publications and software packages

This thesis is based on the following peer-reviewed publications and software packages:

1. **Nguyen, D.A.**, Kong J., Wang H., Menzel S., Sendhoff B., Kononova A.V. & Bäck T.H.W. (2021), Improved automated CASH optimization with tree parzen estimators for class imbalance problems. In IEEE 8th international conference on data science and advanced analytics (DSAA), pp. 1-9, DOI: 10.1109/DSAA53316.2021.9564147.

    - Github: `https://github.com/ECOLE-ITN/NguyenDSAA2021`

2. **Nguyen, D.A.**, Kononova A.V., Menzel S., Sendhoff B. & Bäck T.H.W. (2021), Efficient AutoML via combinational sampling. In IEEE Symposium Series on Computational Intelligence (SSCI). pp. 01-10, DOI: 10.1109/SSCI50451.2021.9660073.

    - Github: `https://github.com/ECOLE-ITN/NguyenSSCI2021`
    - Pypi: `https://pypi.org/project/BO4ML`

3. **Nguyen, D.A.**, Kononova A.V., Menzel S., Sendhoff B. & Bäck T.H.W. (2022), An Efficient Contesting Procedure for AutoML Optimization, in IEEE Access, vol. 10, pp. 75754-75771, 2022, DOI: 10.1109/ACCESS.2022.3192036.

    - Github: `https://github.com/ECOLE-ITN/NguyenIEEEAccess2022`
    - Pypi: `https://pypi.org/project/DACOpt`

4. **Nguyen, D.A.**, Kononova A.V., Kong J., Jonker, K., Pipard, N., Mooi, J. & Bäck T.H.W. (2023), Automated Machine Learning Using Class Importance Weights For Imbalanced Multi-class Classification Of Steel Coil Defects, in review.

5. **Nguyen, D.A.**, Kononova A.V., Kong J., Jonker, K., Pipard, N., Mooi, J. & Bäck T.H.W. (2024), Efficient AutoML Optimization for Imbalanced Multiclass Data: A Case Study on Surface Defect Classification in Steel Manufacturing, in review.

    - Github: `https://github.com/anh05/AutoML-Multiclass-Imbalanced`

Other work by the author:

1. Kong J., Kowalczyk W.J., **Nguyen, D.A.**, Bäck T.H.W. & Menzel S., (2019), hyperparameter optimisation for improving classification under class imbalance. In IEEE Symposium Series on Computational Intelligence (SSCI), pp. 3072-3078, DOI: 10.1109/SSCI44817.2019.9002679.

2. Ullah S., **Nguyen, D.A.**, Wang H., Menzel S., Sendhoff B. & Bäck T.H.W. (2020), Exploring dimensionality reduction techniques for efficient surrogate-assisted optimization. In IEEE Symposium Series on Computational Intelligence (SSCI), pp. 2965-2974, DOI: 10.1109/SSCI4 7803.2020.9308465.