



Universiteit
Leiden
The Netherlands

Efficient tuning of automated machine learning pipelines

Nguyen, D.A.

Citation

Nguyen, D. A. (2024, October 9). *Efficient tuning of automated machine learning pipelines*. Retrieved from <https://hdl.handle.net/1887/4094132>

Version: Publisher's Version

License: [Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden](#)

Downloaded from: <https://hdl.handle.net/1887/4094132>

Note: To cite this publication please use the final published version (if applicable).

Automated Machine Learning: An Overview

The term *machine learning* has become highly popular in today's technology and is expanding rapidly. Without realizing it, we use machine learning in our daily lives, such as in self-driving cars [1], medical diagnosis [2], automatic language translation [5], fraud detection [6], and defect detection [49].

Existing AutoML frameworks aim to automatically build the best ML pipeline for an arbitrary ML problem. However, applying AutoML to produce a useful ML product for a real-world problem eventually requires some knowledge of machine learning and software development. We first discuss some important aspects of the life cycle of machine learning development in Section 2.1. Next, this chapter will discuss all the functions of a typical AutoML tool, as shown in Figure 1.1, except for the optimization part, which we will discuss in Chapter 2 due to its significance in this thesis. Specifically, Section 2.2 provides an overview of commonly used approaches for determining the optimal ML pipeline architecture. Section 2.3 discusses various applications of meta-learning in AutoML. Section 2.4 presents the explainable and low-code techniques utilized in AutoML products.

2.1 Life Cycle of Machine Learning Development

The process of building up a machine learning system can be seen as a combination of the *Software Development workflow* [50] and the *Data science workflow* [51], which is shown in Figure 2.1. This workflow contains four main stages:

- *Data preparation* is the starting point for an ML project where the data is collected [52]. This will be discussed in Section 2.1.1.
- *ML pipeline optimization* is handled by the employed optimizer, which will be discussed in Chapter 3. Hence, Section 2.1.2 discusses the topic from the user's view rather than the view of an optimizer provider. In this section,

2. Automated Machine Learning: An Overview

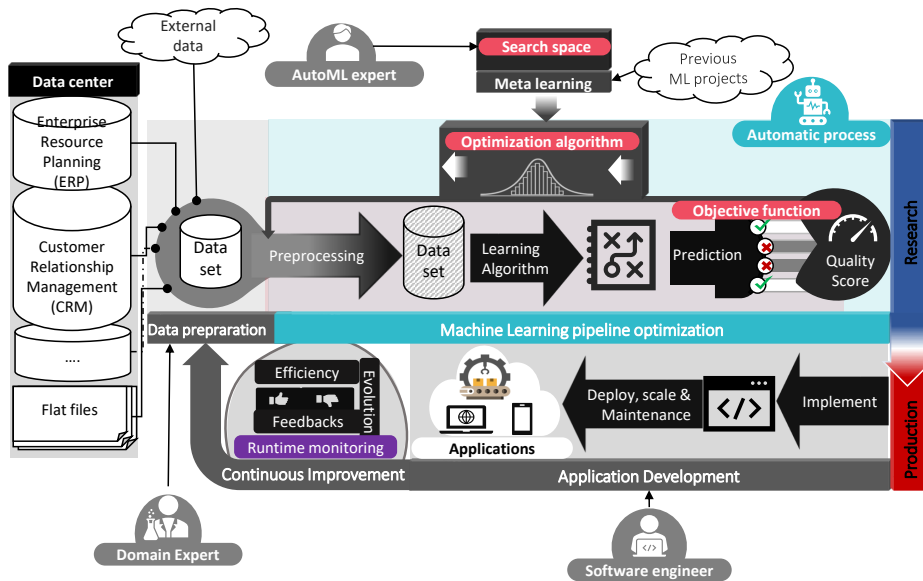


Figure 2.1: Machine learning development life cycle. This figure takes inspiration from the software development [50] and data science [51] lifecycles.

we clarify what parts of an ML pipeline can be applied for optimization for automatic purposes (Section 2.1.2.1), how we can evaluate an ML pipeline (Section 2.1.2.2), and the *over-fitting* problem [53] in Section 2.1.3, a common problem when applying optimization to AutoML [54].

- *Application Development* stage refers to the application development progress, where the final application is produced [50]. An engineering team typically does this stage with expertise in developing applications (e.g., software, web app, embedded application for Computer Numerical Control (CNC) machines).
- *Continuous Improvement* indicates the process of making small incremental changes to the developed application [55]. The ML application development is not a one-off process. In the ML applications development scenario, the constant improvement paradigm is more necessary since the data continues to be updated. The data distribution changes make the ML model's prediction performance less accurate. Thus, the deployed ML model must be retrained to adapt to the updated data. Besides, the selected ML pipeline may no longer be the best choice for the new data. Therefore, the ML pipeline has

to be re-tuned. We refer the interested reader to the Kaizen model [55] for this topic.

2.1.1 Data preparation

In practice, the data used for a particular machine-learning project can be collected from multiple sources, such as Enterprise Resource Planning (ERP), customer relationship management (CRM), network log files, Microsoft Excel, Microsoft Word, images, expert suggestions, or external sources. However, a real-life system, such as an ERP, can have thousands of data tables with complex relationships. Therefore, choosing the tables and data to use is a challenging question. Additionally, data from multiple sources (e.g., data crawled from websites, extracted from multiple internal systems, or gathered end-user working data files) might be incorrectly labelled or inconsistent in identity, used metric system units (e.g., millimetre (mm), centimetre (cm), meter (m)), and format (e.g., DateTime). Collecting data and joining different data sources are important and challenging. Hence, these tasks are typically performed manually by experts who deeply understand the data. In general, data preparation [52], [56]–[58] typically involves the following tasks:

1. **Data collection:** Useful data is obtained from operating systems, data warehouses, data lakes, and other information sources. During this step, it is crucial for data scientists, domain experts, members of the ML team, and end-users to verify that collected data is a good fit for the objectives of the anticipated ML applications [56].
2. **Data discovery and manual cleansing:** This step consists in investigating the gathered data to determine what it includes and what needs to be done to make it suitable for intended usage [52]. Next, the detected data flaws and mistakes are fixed to develop comprehensive and accurate data sets. For instance, faulty data is rectified or removed, missing values are filled in, and inconsistent entries are merged as part of the data sets clean up [59].
3. **Data organizing:** At this stage, the data must be modeled and organized to be suitable for ML [56]. For example, the data stored in comma-separated value (CSV) files and reorganizing image files.

It is worth noting that the target outcomes of all the above steps should include data and a set of prepared rules. The rule can be in the form of hard-coding, workflows, or generic formulas, which can be reused for future data in the production phase.

2. Automated Machine Learning: An Overview

Apart from the mentioned *manual steps*, there are two classes of data-cleaning approaches:

1. *Semi-automatic* is a class of algorithms/tools that assist scientists in improving efficiency and reducing human effort during the data preparation phase, rather than being completely automatic. For instance, KARATA [60] is a tool that uses knowledge-based [61] and crowd-powered [62] approaches, detecting both correct and incorrect data to generate possible repairs for the identified incorrect data. Another class of methods, as exemplified by Krishnan et al. [63]–[65], involves suggesting cleaning for only a limited portion of the data while maintaining comparable outcomes to cleaning the entire dataset. However, these methods require humans to design data-cleaning operations applied to the dataset.
2. A small class of *automatic data cleaning* techniques aims to improve the data quality automatically. It can be applied to various datasets and used in AutoML frameworks (see Section 2.1.2.1 for an additional discussion). Nevertheless, they are usually hard-coded and limited to a few specific functions, such as handling data errors, missing values, redundant records, invalid values, and outliers [22], [66], [67].

2.1.2 Automated Machine Learning Pipeline

This section’s topic is limited to supervised machine learning fields. This section focuses on processes that do not involve humans. We first start by listing the elements of a typical ML pipeline (Section 2.1.2.1). Second, we present the standard evaluation measurement metrics (Section 2.1.2.2). The last sub-section presents the common over-fitting problems when optimization approaches are applied (Section 2.1.3).

2.1.2.1 Machine learning pipeline

A generic ML pipeline $p : \mathbb{X} \rightarrow \mathbb{Y}$ designed for problem solving P is a sequence of operators that transforms a vector of features $\mathbf{x} \in \mathbb{X}$ into a target value $y \in \mathbb{Y}$ which can be, for example, a predicted value for a regression problem or a label for a classification problem. Examples of possible pipeline operators depend on problem P and can include data pre-processing, encoding, feature selection, and resampling.

In order to analyze this discussion, we need to use the notations introduced and explained in Chapter 1. These notations are crucial for our ongoing analysis, and we discussed them in detail in their original context in Chapter 1 to ensure a better understanding. Let $\mathbb{O} = \mathcal{O}_1 \times \dots \times \mathcal{O}_z$ 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} : $p(\mathbf{x}) = \mathcal{O}_z(\mathcal{O}_{z-1}(\dots(\mathcal{O}_1(\mathbf{x}))\dots))$. Each operator's functionality can typically be delivered by one of the multiple available ML algorithms: here we assume $\mathcal{O}_{i \in \{1, \dots, z-1\}} = \{\emptyset, \mathcal{A}_i^1, \dots, \mathcal{A}_i^{n_i}\}$ for all operators except the last and $\mathcal{O}_z = \{\mathcal{A}_z^1, \dots, \mathcal{A}_z^{n_z}\}$ for the last operator that 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 \emptyset .

ML pipeline structure Although several AutoML frameworks have been released to date, there are no best practices for ML pipeline structures for all ML problems in the literature. We only know that the learning algorithm must be at the end of the pipeline. Thus, an AutoML framework creates an ML pipeline using either a fixed structure based on the creator's expertise or a variable structure (the detailed discussion on *ML pipeline structure search* is given in Section 2.2). We note that the number of operators is flexible, as it highly depends on the ML problem (i.e., classification, regression), the underlying domain, and the input data itself (e.g., image, text, and tabulator input require different preprocessing algorithms).

The considered operators are usually grouped into two main phases:

1. **Preprocessing phase** includes several preprocessing tasks that can be seen as an augmentation step adding to the data preparation phase, but it is automatic. This step includes a sequence of optional steps. For example, a typical preprocessing sequence for a classification problem includes missing-value imputation, categorical encoding, data normalization, resampling, feature extraction, feature generation, and feature selection. Generally speaking, any data modification before utilizing a learning method is referred to as the *preprocessing step*. Note that the input data might be changed sequentially after processing via preprocessing operators.
2. **Learning phase** is the last operator in the ML pipeline. It aims to learn the relation within the dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$ output from previous operator \mathcal{O}_{z-1} and able to predict those of unseen dataset $\mathcal{D}_{\text{unseen}} =$

2. Automated Machine Learning: An Overview

$\{(\mathbf{x}_{m+1}, y_{m+1}), \dots, (\mathbf{x}_{m+t}, y_{m+t})\}$. More specifically, the learning algorithm aims at finding a learning function $f : \mathbb{X} \rightarrow \mathbb{Y}$ by generalizing beyond \mathcal{D} , that maps inputs $\mathbf{x} \in \mathbb{X}$ to outputs $y \in \mathbb{Y}$, i.e., $y = f(\mathbf{x})$. The learning function f can be formed by any option from the set of possible learning algorithms, e.g., linear models [68]–[70], tree-based models [71], [72], support vector machines [21], k -nearest neighbors [20], etc.

2.1.2.2 Evaluation measurement metrics

Finding the optimal ML model for the target problem is the main task in ML pipeline optimization. Hence, the vital questions are: How well does the model perform? Moreover, what is the accuracy of the model? To do so, we need a *performance metric* (or evaluation measure) to score the model's quality. The performance metric depends on the target problem, i.e., a classification problem or regression problem. For instance, the *accuracy rate* is usually used in classification problems [73]. However, when the classes are imbalanced, the *geometric mean* (GM) and *F-measure* are highly recommended [47], [74], [75] because of their ability to represent the minority class samples. In contrast, the *Mean Absolute Error* (MAE), and *Mean Squared Error* (MSE) are typically used to score a regression model [76]. Hence, choosing a suitable performance metric is the task given to experts. In this section, we will present the most commonly used performance metrics for classification and regression problems. For unsupervised machine learning, we refer the interested reader to other reviews of evaluation metrics for further discussions on that domain [77].

Measurement metrics for classification problems Classification algorithms stand for algorithms that predict label $y \in \mathbb{Z}$, i.e., discrete/categorical value such as Spam/Not Spam in Email Spam Filtering problem [78], based on a vector of features $\mathbf{x} \in \mathbb{X}$. Therefore, the accuracy of an individual prediction is either correct or incorrect by comparing the predicted value and the actual value. By comparing the predicted and actual values, the accuracy of an individual prediction is either correct or incorrect. The accuracy rate is most commonly used, as it simply computes the ratio of the number of samples correctly predicted to the total number of tested samples. However, the accuracy rate is not the sole metric to evaluate a classification model. In this section, we will summarize a total of seven performance metrics that are usually used for classification problems. This discussion is restricted to *binary-classification* problems. We refer the interested

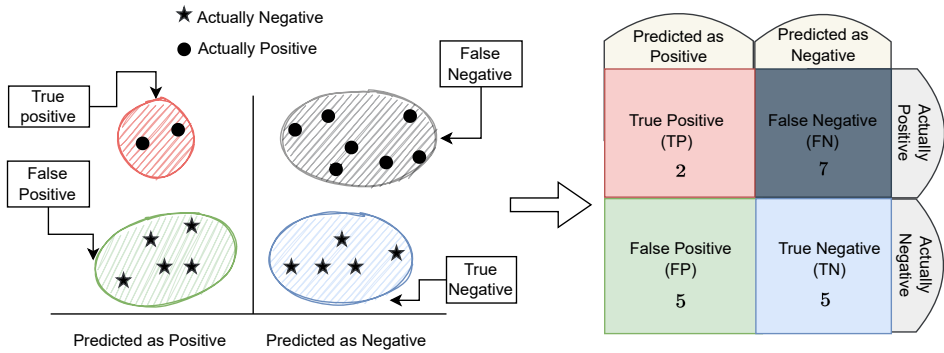
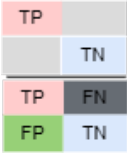
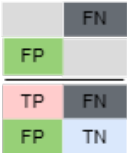

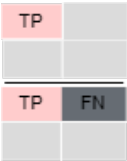
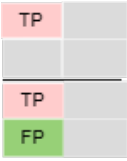


Figure 2.2: Illustration of confusion matrix. The illustration is inspired by the work of [79].

reader to Chapter 6 (Section 6.2.2) for multi-class classification problems and other reviews of evaluation metrics for other types of classification problems [73], [75], [80]. For binary classification problems, the performance metrics can be defined based on the confusion matrix. For consistency throughout this section, we use the confusion matrix example shown in Figure 2.2, as it can provide intuitive classification results. The figure on the left shows the distributions of the predicted and actual classes. The plot on the right shows a confusion matrix for this example. Using the confusion metric in Figure 2.2, we summarized several commonly used performance metrics in Table 2.1, including Accuracy rate, Error Rate, Specificity (or True Negative Rate), Sensitivity (or Recall or True Positive Rate), Precision, Balanced accuracy, Geometric mean¹, F_β -measure.

¹The geometric mean mentioned here follows the work of [81], which is based on Sensitivity (accuracy on positive examples) and Specificity (accuracy on negative examples). Some other studies might be based on precision and recall.

2. Automated Machine Learning: An Overview

Name	Formula	Illustration	Referring to Figure 2.2	Description
Accuracy rate	$\frac{TP+TN}{TP+FP+TN+FN}$		$\frac{2+5}{2+5+5+7} = 0.37$	The ratio of correct predictions on tested samples
Error Rate	$\frac{FP+FN}{TP+FP+TN+FN}$		$\frac{5+7}{2+5+5+7} = 0.63$	The ratio of incorrect predictions on tested samples
Specificity /True Negative Rate (TN_{rate})	$\frac{TN}{TN+FP}$		$\frac{5}{5+5} = 0.5$	The ratio of the number of correctly predicted negative samples overall actual negative samples
Sensitivity /Recall /True Positive Rate (TP_{rate})	$\frac{TP}{TP+FN}$		$\frac{2}{2+7} = 0.22$	The ratio of the number of correctly predicted positive samples overall actual positive samples
Precision	$\frac{TP}{TP+FP}$		$\frac{2}{2+5} = 0.29$	The ratio of actual positive samples among those predicted as positive

continued on the next page

Table 2.1: Performance Metrics for Classification Evaluations.

Name	Formula	Referring to Figure 2.2	Description
Balanced accuracy	$\frac{\text{Specificity} + \text{Sensitivity}}{2}$	$\frac{0.5 + 0.22}{2} = 0.36$	The arithmetic mean of class-wise sensitivity, i.e., Specificity and Sensitivity
Geometric mean	$\sqrt{\text{Specificity} \times \text{Sensitivity}}$	$\sqrt{0.5 \times 0.22} = 0.33$	The root of the product of class-wise sensitivity
F_β - measure	$F_\beta = (1 + \beta^2) \frac{\text{precision} \times \text{recall}}{\beta^2 \text{precision} + \text{recall}}$		The weighted harmonic mean of the precision and recall .
	$F_1 = (1 + 1^2) \frac{\text{precision} \times \text{recall}}{1^2 \text{precision} + \text{recall}}$	$2 \times \frac{0.29 \times 0.22}{0.29 + 0.22} = 0.25$	$\beta = 1$ is typically used (i.e., F_β becomes F_1), also means the recall and the precision are equally important. Otherwise, recall is considered β times as important as precision .

Table 2.1: Performance Metrics for Classification Evaluations – continued from previous page

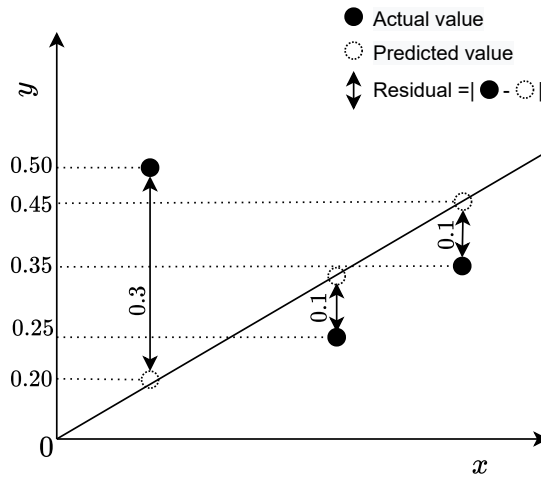


Figure 2.3: Example of regression prediction. The illustration is inspired by the work of [84].

Measurement metrics for regression problems Regression algorithms, as opposed to classification algorithms, predict a specific numeric value, i.e., $y \in \mathbb{R}$. For instance, weather forecasting [82] and housing price forecasting [83]. Hence, the accuracy of a prediction becomes the residual between the predicted value \hat{y} and the actual value y , i.e., $\text{residual} = |y - \hat{y}|$ (see Figure 2.3 for illustration). The four common measurement metrics for regression problems are summarized in Table 2.2. For consistency reasons throughout this sub-section, we use the example shown in Figure 2.3, e.g., each method will be calculated base on that example. In addition, notations are shared throughout this section: t denotes the total number of tested samples.

Name	Formula	Referring to Figure 2.3	Description
Mean Abso- lute Error (MAE)	$\frac{1}{t} \sum_{i=1}^t y_i - \hat{y}_i $	$\frac{0.3+0.1+0.1}{3} = 0.17$	The average of the absolute differences between the actual values and the predicted values.
Mean Squared Error (MSE)	$\frac{1}{t} \sum_{i=1}^t (y_i - \hat{y}_i)^2$	$\frac{0.3^2+0.1^2+0.1^2}{3} = 0.04$	The average of the squared differences between the actual values and the predicted values.
Root Mean Squared Error (RMSE)	$\sqrt{\frac{1}{t} \sum_{i=1}^t (y_i - \hat{y}_i)^2}$	$\sqrt{0.04} = 0.2$	The square root of MSE
R^2 score	$1 - \frac{\sum_{i=1}^t (y_i - \hat{y}_i)^2}{\sum_{i=1}^t (y_i - \bar{y})^2}$ $\bar{y} = \frac{1}{t} \sum_{i=1}^t y_i$	$\bar{y} = \frac{0.5+0.25+0.35}{3} = 0.37$ $R^2 = 1 - \frac{0.11}{0.03} = -2.67$	The coefficient of determination

Table 2.2: Performance Metrics for Regression Evaluations.

2. Automated Machine Learning: An Overview

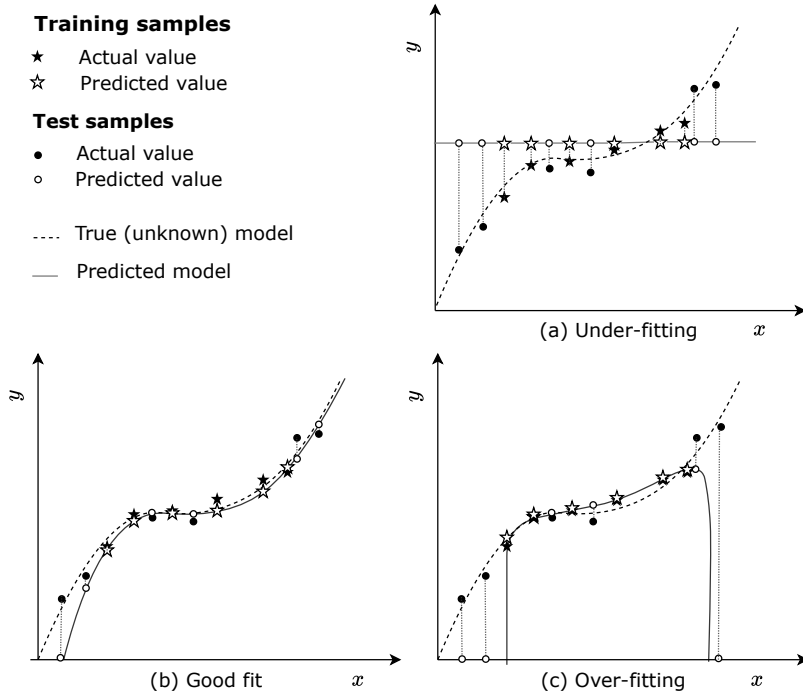


Figure 2.4: Illustration of Under-fitting (a), Good fit (b), and Over-fitting (c). The illustration is inspired by the work of [85].

2.1.3 Over-fitting and under-fitting

Generalization is a common problem in AutoML and hyperparameter optimization[53], [86]. During the optimization process, the highest-performing configuration is discovered through multiple trials on a training dataset. However, this optimal configuration may not generalize well to new, unseen data, leading to what is known as the over-fitting problem. Therefore, addressing over-fitting is an important concern in AutoML optimization. This section will provide an overview of the overfitting problem and highlight common practices to avoid overfitting in AutoML. In supervised machine learning, we have to find the best form of the function f that minimizes the difference between true value y_i and predicted value \hat{y}_i , i.e., $y_i \simeq f(\mathbf{x}_i), \forall \mathbf{x} \in \mathbb{X}$. Hence, the learning algorithm aims to produce a model that fits the data. However, if the model is *too fit* to the training data, it may result in the so-called *over-fitting problem*, in which the model performs well on the known training set but performs poorly on the unknown test set. In other words, the model does not have *generalization ability*. For clarification, Figure 2.4

2.1 Life Cycle of Machine Learning Development

illustrates under-fitting, good fit, and over-fitting problems. Assuming we have 12 samples, the 6 stars denote training samples, the 6 dots are test samples, the black colour indicates actual values, and the white colour indicates predicted values. The dashed curved line indicates the actual model, and the solid curved line indicates the predicted model. The plot in (a) shows an example of an underfitting. We can see that the predicted and actual models are completely different; the performance is poor for both the training and test samples. The predicted model in plot (b) was similar to the actual model. In this case, the predicted model may be a good fit. The last case is an example of over-fitting. The predicted model perfectly models the training samples but lacks generalization ability. Consequently, the model yielded poor predictions for the test samples. Under-fitting can easily be detected via performance metrics (see Section 2.1.2.2).

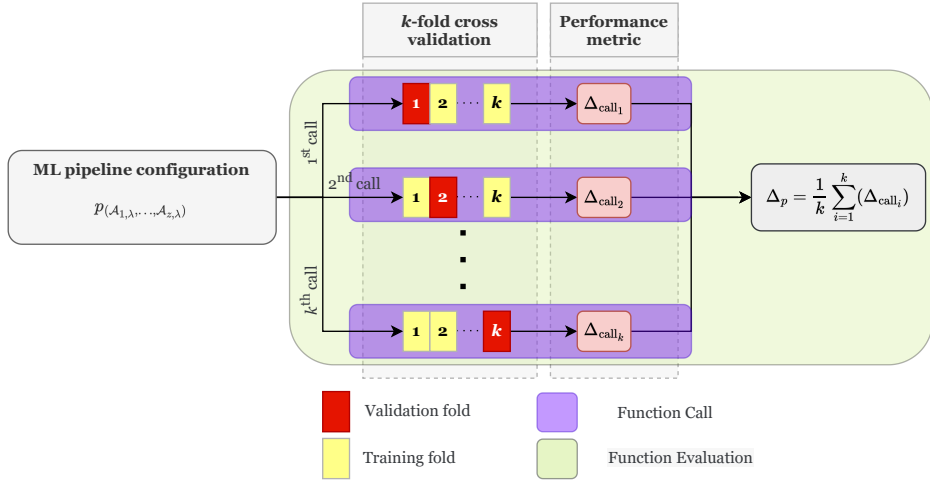


Figure 2.5: k -fold cross validation.

A common technique for avoiding over-fitting is k -fold cross validation (see Figure 2.5) in which the input data is randomly partitioned into k independent folds (also called *leave-one-out* technique). The average of the k function calls, computes the performance of one test case; a single fold is kept as a test set, while the remaining folds are used as the training set. The term *function evaluation* (green rounded-box) and *function call* (purple rounded-box) will be used in future discussions in this thesis, e.g., Chapter.3. Aside from k -fold cross-validation, there are several other techniques for preventing over-fitting, such as early stopping [68],

2. Automated Machine Learning: An Overview

regularization [87], [88]. We refer the interested reader to those studies for further discussions on this topic.

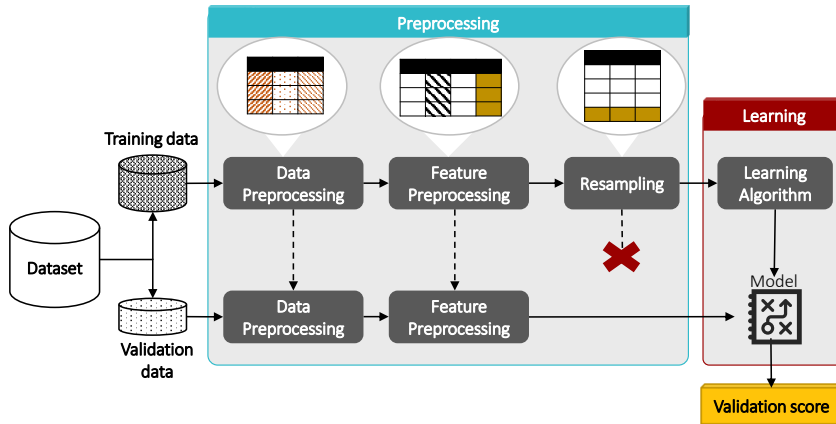


Figure 2.6: An example of calculating the validation performance of a typical Machine Learning pipeline.

Finally, when training and validating a model, we usually use exactly the same ML pipeline architecture for preprocessing and learning phases, for example, encoding categorical data and normalizing data in the same way. However, some operators in the preprocessing step must not be applied to the validation/test data. Otherwise, all techniques for overcoming over-fitting are useless. In general, the preprocessing can be grouped into two groups:

- Impact on the individual sample's quality, e.g., missing value imputation, categorical encoding, normalization data, feature extraction, feature generation, and feature selection.
- Changing the original distribution of the samples, for example, resampling (e.g., under-resampling, over-resampling, combine-resampling techniques) and removing outliers.

The first group has to stick to applying precisely the same (i.e., algorithms, hyperparameter settings, and their order) to both the training and test/validation sets. In contrast, the second group impacts the sample size to improve the training model's generalization and quality. Recall that a model's test/validation performance is based on the prediction values and actual values of the entire test/validation dataset samples — no more or fewer samples. Hence, the quality

measurement is incorrect when we base it on more or fewer samples' predictions than the actual test/validation set.

In other words, no approach that impacts the integrity of samples can be applied to the test/validation dataset. Figure 2.6 shows a workflow of a typical ML pipeline when applied to the training and validation data.

2.2 ML pipeline architecture search

The first step in solving any ML problem is to find a suitable machine learning pipeline structure. An ML pipeline includes several optional operators (e.g., imputation of missing data, encoding, scaling, feature extraction, and feature selection) and a mandatory learning operator. The generic ML pipeline is shown in Figure 2.7. The ML pipeline architecture search aims to answer the following research question: How many operators are required in the pipeline? Moreover, how are they ordered? The only known is the last operator, which is a learning algorithm, that is, classification, regression, or clustering. However, there is no fixed rule in the early steps, which confuses non-experts when creating their own ML pipelines.

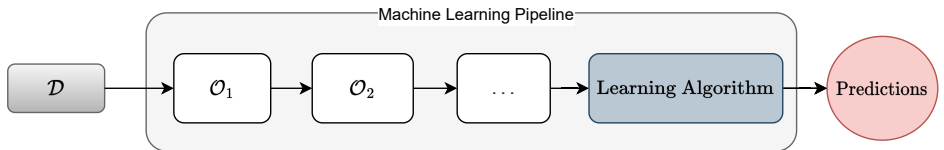


Figure 2.7: Prototypical ML pipeline architecture.

In addition, the performance of an ML pipeline is calculated based on its prediction quality on test data, that is, we only evaluate an ML pipeline when the input data are past all pipeline operators. Finally, every ML pipeline is unique [15], i.e., any changes to the ML pipeline lead to a different ML pipeline, which might change the final performance.

2.2.1 Fixed ML pipeline architecture

Many AutoML frameworks, such as Auto-Weka [40], [41], Hyperopt-sklearn [42], H2O [89], ATM [90], Auto-Gluon [91], [92] do not directly solve ML pipeline architecture search to reduce the complexity of determining ML pipeline structure.

2. Automated Machine Learning: An Overview

Instead, they have predefined a fixed structure, which closely resembles a well-known linear sequence of operators recommended in literature or based on their experiences. Figure 2.8 shows an example of a commonly used fixed ML pipeline architecture for class-imbalanced problems, as used by [15], [37], [47], [74], [75]. A possible disadvantage of this approach is that it might result in inferior predictive performance for complex datasets requiring, e.g., multiple preprocessing steps.

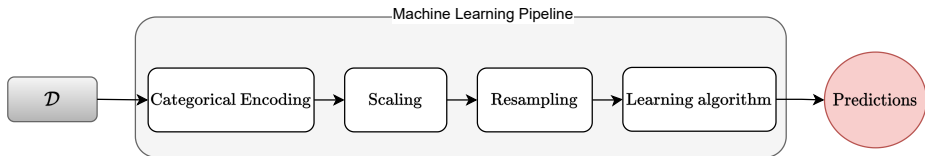


Figure 2.8: Fixed ML pipeline architecture used by most imbalanced-class classification studies.

2.2.2 Flexible ML pipeline architecture

In order to achieve the best performance for a given problem, human experts usually build highly specialized ML pipelines, i.e., the ML pipeline is adaptable to a specific task. An illustration on ML pipeline architecture search is shown in Figure 2.9.

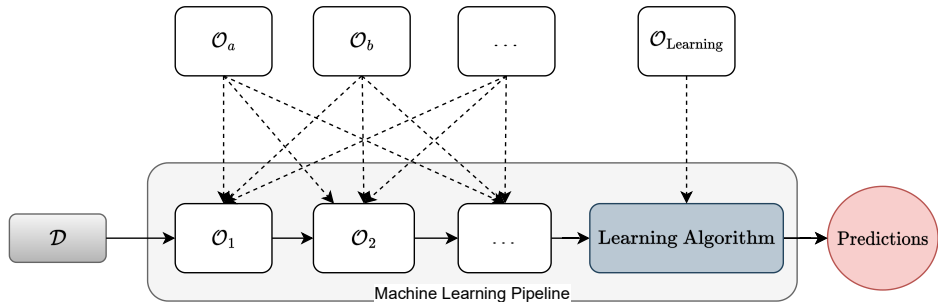


Figure 2.9: Flexible ML pipeline architecture, where the ML pipeline structure is also searched for.

This flexibility is missing from the fixed ML pipeline structure approach. To address this issue, one class of AutoML frameworks considers flexible ML pipeline architecture. AUTO-SKLEARN [39], [45] can be seen as a semi-flexible approach. AUTO-SKLEARN predefined a set of fixed structures for ML problems. For a given ML problem, it generates a pipeline structure using meta-features of the input

AutoML tool	Underlying technique domain
TPOT	Genetic programming [102], [103]
ALPHAD3M	Reinforcement learning [104]
MLPLAN	Hierarchical task networks [105]
MOSAIC	MonteCarlo tree search [106], [107]
FEDOT	Evolutionary algorithms [108]–[110]

Table 2.3: AutoML frameworks support ML pipeline architecture search

data, e.g., the structure for binary and multi-class classification problems might be different. However, the same structure will be used for the same ML problem domain. Apart from that *if-else* style, there is a class of AutoML frameworks that support ML pipeline architecture search, including TPOT [43], ALPHAD3M [93], [94], ML-PLAN [95], [96], and P4ML [97], MOSAIC [98], FEDOT [99]. The main idea of these approaches is to apply restrictions in the form of ad hoc configuration, primitive taxonomies, or context-free grammars [100], [101]. Table 2.3 provided the relevant techniques to these AutoML tools.

Lastly, a class of techniques based on meta-learning will be discussed in Section 2.3.

2.3 Meta Learning

A typical ML pipeline optimization problem consists of three fundamental components – a search space, an optimizer, and an objective function. The search space describes the feasible search domain. The optimizer is used to discover the best combination of algorithms over operators and their optimized hyperparameters, thereby maximizing the performance of the objective function. Finally, the objective function is a child program that evaluates the settings of the ML pipeline, resulting in a real-valued performance measures, such as accuracy, precision, and recall rate. In other words, the ML pipeline optimization process aims to find the most suitable solution from a predefined search space.

Usually, for optimizing a new (unknown) ML problem, the optimizer explores the search space from scratch. In contrast, ML experts take advantage of previous tasks (e.g., referring to literature and their experiences so far) to shorten the optimization process and avoid wasting time on unpromising search areas. Inspired by the behavior of human experts when dealing with a new ML task, meta-learning is quite similar to learning from the experiments of previous ML tasks to increase

2. Automated Machine Learning: An Overview

the efficiency of the search. Meta-learning is the science of learning similar datasets, which can be characterized by a set of *meta-features* [111], which might include:

1. *Simple features* includes the following: number of samples, number of features, number of classes, number of missing values, number of instances with missing values, number of numeric features, number of categorical features, number of binary features.
2. *Statistical features*: mean, standard deviation, mean skewness, quarterlies.
3. *Information-theoretic features*: class entropy, mean mutual information.
4. *Land-marking* [112]: Performance evaluations of some simple classifiers on the entire data or sub-set of data [113], e.g., performances evaluated by k-nearest neighbor with 1 neighbor.

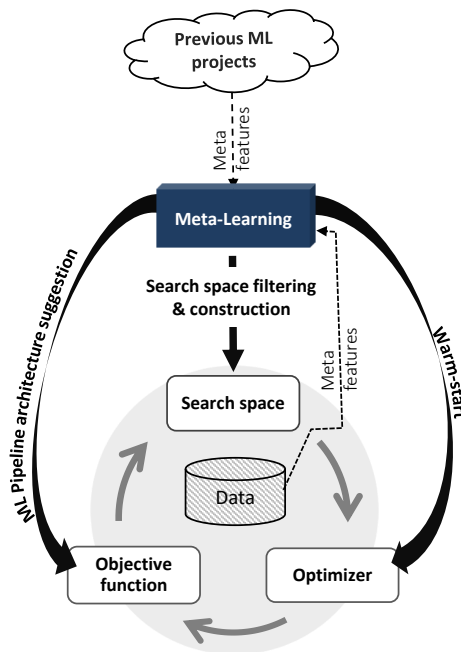


Figure 2.10: Applications of meta-learning. The illustration is inspired by the work of [114], [115].

Meta-learning can be applied in many stages of the ML pipeline optimization process (see Figure 2.10):

- *Search space filtering and reduction:* The search space is usually predefined by the AutoML owners, which is typically large with many possible algorithms that have been integrated. Furthermore, the hyperparameters of the chosen algorithms are set by a wide range of values. Nevertheless, the search space is arbitrarily defined without referencing the given ML problem. Therefore, meta-learning can be used to reduce the large search space, such as eliminating unimportant hyperparameters, irrelevant algorithms (e.g., [116]–[121]), and automatically construct a (minimal) search space (e.g., [101], [122]).
- *ML pipeline architecture suggestion:* Apart from the mentioned approaches for flexible ML pipeline architecture in Section 2.2. Meta-learning is a promising research domain for identifying ML pipeline synthesis. [101] proposed a data-centric approach, called DSWIZARD, that learns from related ML tasks to construct a suitable ML pipeline for the target problem. Similarly, predicting the pipeline’s performance and favoring a good pipeline architecture to be constructed was studied in [94], [97], [98], [111], [123], [124].
- *Warm-start for optimizer:* Traditionally, the optimization process often starts from scratch. For example, the initialization step in Bayesian optimization (Section 3.1.3) randomly selects some ML pipeline configurations without any evaluation of the given dataset, that is, with the same random seed – the optimizer generates the same set of initialization configurations for any dataset. On the other hand, by learning from previous tasks, many studies proposed to start from a set of the best configurations of the related tasks [39], [120], [125]–[127]. In this manner, the underlying optimizer can characterize the search space by focusing on promising areas for the given data. Consequently, the optimizer maximizes the chance of finding the best solution early. This is the general idea of a warm-start in ML pipeline optimization.

Finally, we refer the interested reader to other reviews of meta-learning for further discussions on this topic [114], [115].

2.4 Explainable and low-code for AutoML

AutoML and optimization studies have been applied in many industrial domains [7], [128]–[130]. However, the current state of AutoML is only seen as a reference tool for human experts in real-life application development [131]. From a practitioner’s

2. Automated Machine Learning: An Overview

perspective, AutoML is also seen as a black-box, i.e., optimization process, that optimizes another black-box, i.e., objective function [11]. Also, different frameworks might lead to different results and suggestions. Many studies have been noted that users do not trust AutoML systems [12], [132]–[134]. Without understanding its optimization behaviors, users might not confidently decide to use its suggestion (i.e., the best-found ML pipeline). For instance, [12] has revealed that even though AutoML might deliver high-quality solutions, practitioners refuse to use them as they do not want to be held accountable for a model they do not understand. In addition, [11] concluded that the main reason for the limited trust of practitioners is the limited explanation and transparency of the outcome of AutoML.

Therefore, establishing trust in an AutoML system is an important motivation for explainability [135], that is, explaining in a way that humans can understand in a reasonable time [136].

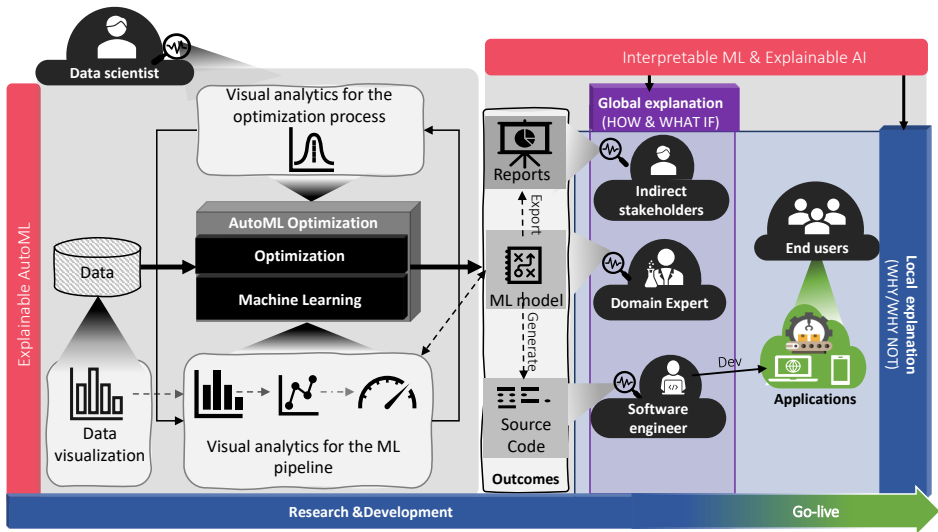


Figure 2.11: Requirements analysis of different user groups on their need for ML explainability. Explainable AutoML is a sub-class of explainable AI that targets ML high-skill user groups that can read/understand ML algorithms and visualization techniques.

2.4.1 Stakeholders of AutoML

Stakeholders involved in a ML project can be grouped into three groups:

- *Direct stakeholders*: Indicate actors who are directly involved in the whole process of developing their final ML application with the day-to-day activities, e.g., data scientists, domain experts, and software engineers.
- *Indirect stakeholders*: People are not affected by the project but may support the project in some terms but not focused on the process of finishing it, such as system infrastructure administrators who may prepare a suitable system infrastructure and system configuration, legal advisors might be involved in the term of making the end-users policies, managers who set the business goals for the project, build the project team, customers (for an internal project, and the customers can be the board of managers) might approve the project results.
- *End-users*: They can be internal or external actors who will use the output application for their daily work.

2.4.2 Components of an explainable AutoML

Each group of actors had different skill sets, knowledge, and demands. Thus, the explanation is that adaptability is based on its properties. Figure 2.11 summarizes the requirements analysis of different user groups on their need for ML explainability. The co-badged guidance [137] defines the explainability of intelligent systems as a combination of technical (information extraction) and non-technical (communication method) considerations. Furthermore, recall that AutoML is a combination of optimization and ML. Hence, the scope of explainable AutoML includes both aspects. According to the relevant discussions on optimization [138] and machine learning [51], we formulate the technical explainability requirements for AutoML as three complementary approaches that form to increase trust and transparency:

1. *Global explanations on the optimization level* aim to explain the decision-making process of the optimizer. In addition, this level of explainability provides helpful information on optimizer behavior to illustrate optimization convergence and how it constructs the pipeline.
2. *Global explanations on a particular model level* aim to explain the general model’s decision-making process. This level of explainability is about understanding *how* the model makes decisions and the distribution of the target outcome based on the features.

2. Automated Machine Learning: An Overview

3. *Local explanations for a single prediction*: Global explainability is more useful in the research and development phase as it helps project owners determine how their data distribute and how the ML pipeline transforms the input data to lead to the final result. The local explainability helps examine what the model predicts for a particular sample and explains *why/ why not*.

Those applications of explanation are used for many demands, such as debugging ML models [139], explaining medical decision-making [140], explaining predictions for classification problems [141], and explain autonomous agent behavior [142], [143].

2.4.3 Maturity Levels of Automation Tools

According to Figure 2.11, the optimization and programming phases necessitate a high level of technical knowledge. The optimization phase requires data science skills, and the programming phase aims to build the deliverable application, which requires coding skills. Both target a goal that can be used by the layperson (lay scientist/ developer). The levels of automation tools for AutoML are summarized in Figure 2.12.

2.4.3.1 Tools for data scientist

To be able to understand and be accountable for the outcome of AutoML in a real-world application, the practitioner usually investigates the optimization behaviors and its final suggestions. For example, the practitioner may plot several visualizations of how the data transformed through the pipeline or trace back the optimizer’s decision-making process by plotting the necessary figures that they can explain to others (i.e., non-technical users). However, the process of making figures might be costly, as it depends on how familiar the data scientist is with the platforms used. Hence, the practitioners must develop practice skills in ML and coding [132]. That fact limits the usefulness of AutoML and misses the opportunity to lead toward helping humans apply ML to real-life applications with limited ML and statistics knowledge.

To overcome these limitations and save time for data scientists, some AutoML platforms provide an additional set of visualizations to explain the decision-making process. These can be classified into two main groups:

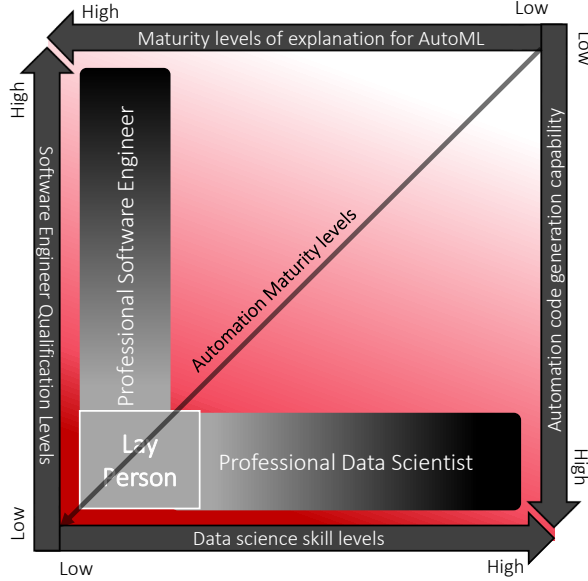


Figure 2.12: Automation Maturity levels for AutoML. The Top-bottom arrows show levels of explainability vs. the data science skill levels that can understand the corresponding explanation related to data scientists. The left-right arrows indicate the levels of automation in building an ML application vs. the needed skill.

1. *Command line based (CLB)*: This set of AutoML platforms provides visualizations via built-in functions. The user can plot figures by executing those functions. However, the user must have a ready-to-use development tool, such as Jupyter Notebook, Anaconda, or dot Net Visual Studio, as well as basic coding skills (e.g., Python, R, C#). They mainly focus on satisfying the *global explanations of the optimization level*. Their functions provide ways to visualize historical data over the tuning process regarding the performance (e.g., [45], [144]), the optimizer behavior for choosing hyperparameter values (e.g., [144]–[146]), and comparing ML pipeline structures (e.g., [11], [147]).
2. *Graphical user interface based (GUI)*: refers to a group of platforms that work as standalone software or web apps and is aimed at users who do not have coding skills. All steps, starting from importing the dataset to tuning and visualization, are integrated into a single interface. Well-known platforms are Google Vizier [145], HyperTuner [148], HyperTendril [149], IBM Watson

2. Automated Machine Learning: An Overview

Studio², Microsoft Azure³, Databricks AutoML⁴, Rapid-Miner⁵. They significantly improved development speed. However, their main limitation is the limited range of applicability, manageability, and flexibility than the others.

Lastly, visualization is an important component, but it is only considered a support function in the value chain of AutoML. Therefore, some open-sourced AutoML platforms might not be interested in developing that function. Fortunately, several platforms add visualization functions on top of other platforms to create a new platform that supports explainability well. AUTOVIZ [150], for example, is an extended version of AUTOAI [131], ATMSEER [134] extends ATM [90], and XAUTOML [11] based on five other platforms are – AUTO-SKLEARN [39], [45], DSWIZARD [101], SCIKIT-LEARN [151], FLAML [152] and OPTUNA [144]. While HYPEROPT [153] was used in GOOGLE VIZIER [145], HYPERTUNER [148] and HYPERTENDRIL [149].

2.4.3.2 Tools for software engineer

This section reviews the low-code technique used in AutoML products, mainly supporting the programmer in producing the final ML product rather than the scientist in the research phase. Low-code and no-code refer to software development methodologies that indicate the concept of practitioners creating their solutions with little (or no) technical skills. Technically, the no-code platforms consist of standard pre-built components and a visual development tool that allows practitioners to use a graphical interface to build their application in a *drag-and-drop* (e.g., SWAY AI⁶) or a wizard-based interface (e.g., AKKIO⁷) styles. However, those platforms are typically limited to some predefined problems, such as object detection in computer vision (MAKEML⁸), image classification (LOBE⁹). That is to say, the automation ability (i.e., ready-to-use) is inversely proportional to the range of applicability. Thus, AutoML platforms that target solving unlimited problems usually stop at the basic level of low code as they only generate the relevant source code for the ML model, e.g., Amazon SageMaker Autopilot [154] export ready-to-use Jupyter notebooks for tested ML pipelines.

²<https://www.ibm.com/cloud/watson-studio/autoai>

³<https://azure.microsoft.com>

⁴<https://databricks.com>

⁵<https://rapidminer.com/>

⁶<https://sway-ai.com>

⁷<https://www.akkio.com>

⁸<https://makeml.app/>

⁹<https://www.lobe.ai>