Taking Human out of Learning Applications: A Survey on Automated Machine Learning

Quanming Yao, Mengshuo Wang, Hugo Jair Escalante,
Isabelle Guyon, Yi-Qi Hu, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, Yang Yu

Abstract—Machine learning techniques have deeply rooted in our everyday life. However, since it is knowledge- and labor-intensive to pursuit good learning performance, human experts are heavily engaged in every aspect of machine learning. In order to make machine learning techniques easier to apply and reduce the demand for experienced human experts, automatic machine learning (AutoML) has emerged as a hot topic of both in industry and academy. In this paper, we provide a survey on existing AutoML works. First, we introduce and define the AutoML problem, with inspiration from both realms of automation and machine learning. Then, we propose a general AutoML framework that not only covers almost all existing approaches but also guides the design for new methods. Afterward, we categorize and review the existing works from two aspects, i.e., the problem setup and the employed techniques. Finally, we provide a detailed analysis of AutoML approaches and explain the reasons underneath their successful applications. We hope this survey can serve as not only an insightful guideline for AutoML beginners but also an inspiration for future researches.

Index Terms—automatic machine learning, neural architecture search, hyper-parameter optimization, meta-learning, transfer-learning

arXiv:1810.13306v1 [cs.AI] 31 Oct 2018

1 Introduction

At the beginning of Tom's famous machine learning textbook [1], he wrote: "Ever since computers were invented, we have wondered whether they might be made to learn. If we could understand how to program them to learn - to improve automatically with experience - the impact would be dramatic". This question gave birth to a new research area, i.e., machine learning, for the computer science decades ago, which tries to program computers improving with experience. Till now, machine learning techniques have been deeply rooted in our every day's life, such as recommendation when we are reading news and handwriting recognition when we are using our cell-phones. Furthermore, it also has got significant achievements in academy, especially in recent years. For example, AlphaGO [2] beat human champion in the game of GO, ResNet [3] surpassed human performance in image recognition, Microsoft's speech system [4] approximated human level in speech transcription.

However, these successful applications of machine learning are far from automation. Since there are no algorithms can achieve good performance on all possible learning problems with equal importance, every aspect of machine learning applications, such as feature engineering, model selection, and algorithm selection (Figure 1), needs to be carefully configured, which is usually involved heavily with human experts. As these experts are rare resources and their designs and choices in above aspects are not replicable, the above success comes at a great price. Thus, automatic machine learning is not only a academic dream described in Tom's book, but is also of great practical usage. If we can take human out of these machine learning applications, we can enable fast deployment of machine learning solutions across organizations, quick validate and benchmark the performance of deployed solutions, and make human focus more on problems depending on applications and business. These would make machine learning much more available for real-world usages, leading to new levels of competence and customization, of which the impact can be indeed dramatic.

Motivated by the above academic dream and practical needs, in recent years, the automated machine learning (AutoML) itself has emerged as a new sub-area in machine learning. Specifically, as illustrated in Figure 1, AutoML attempts to reduce human assistance in the design, selection and implementation of various machine learning tools used in applications' pipeline. It has got increasingly more attention not only in machine learning but also in computer vision, data mining and natural language processing. Besides, AutoML has already been successfully applied in many important problems (Table 1).

TABLE 1
Examples of AutoML approaches in industry and academic.

application	industry	academic
automatic model selection	Auto-sklearn	[5], [6]
neural architecture search	Google's Cloud	[7], [8]
automatic feature engineering	FeatureLab	[9], [10]

The first example is Auto-sklearn [5]. As different classifiers are applicable to different learning problems [1], it is naturally to try a collection of classifiers on a new problem, and then get final predictions from an ensemble of them. However, picking up the right classifiers and setting up their hyper-parameters are tedious tasks, which usually require involvements of humans. Based on the popular

- *Q. Yao, M. Wang, W. Tu and Q. Yang are with 4Paradigm Inc, Beijing, China; H. Escalante is with Instituto Nacional de Astrofisica, Mexico; I. Guyon is with Paris-Saclay (UPSud/INRIA), France and and ChaLearn, US; Y. Hu, Y. Li and Y. Yu are with Nanjing uiversity, Jiangsu, China.*
- *All authors are in alphabetical order of last name (except the first two).*
- *This is a preliminary and will be kept updated, any suggestions and comments are welcome.*
- *Correspondance to Q. Yao at yaoquanming@4paradigm.com*

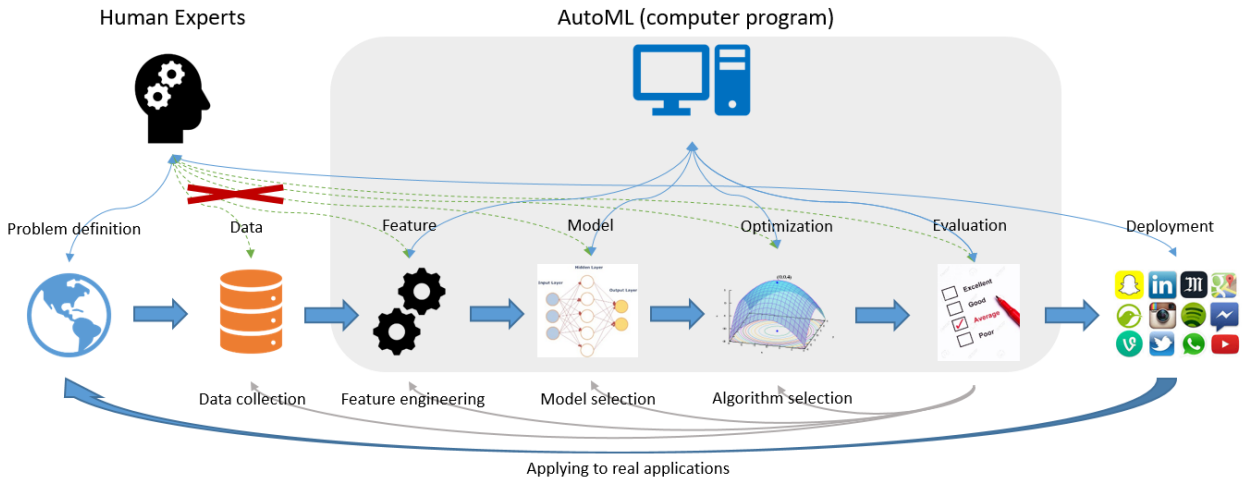


Fig. 1. To use machine learning techniques and obtain good performance, humans usually need to be involved in data collection, feature engineering, model and algorithm selection. This picture shows a typical pipeline of machine learning application, ands how can AutoML involve with the pipeline and minimize participations of humans.

scikit-learn machine learning library [11], Auto-sklearn can automatically find good models from the out-of-the-box machine learning tools for classification. It searches for proper models and optimizes their corresponding hyper-parameters. Thus, it frees humans from above tedious tasks and allows them to focus on the real problem.

The second example is the neural architecture search (NAS) [7], [12], [13]. Since the success of AlexNet [14] on image classification of ImageNet dataset [15], the change on the design of neural architectures has been the main power-source for improving the learning performance. Examples are VGGNet [16], GoogleNet [17], ResNet [3] and DenseNet [18]. Hence, the problem is that can neural architectures be automatically designed so that good learning performance can be obtained on the given tasks. Many researchers have been working on this problem, and reinforcement learning [19] has been developed as a powerful and promising tool for NAS [7], [13], [20], [21]. Besides, NAS has been used in Google's Cloud AutoML, which removes customers from the difficult and time-consuming process of designing architecture.

The last example is automatic feature engineering. In traditional machine learning methods, the modeling performance depends greatly on the quality of features [1]. Hence, most machine learning practices take feature engineering as a vital preposition step, where useful features are generated or selected. Such operations, in the past, are usually carried out manually by data scientists with in-depth domain knowledge in a trial-and-error manner. Automatic feature engineering [9], [10] aims to construct a new features set, with which the performance of subsequent machine learning tools can be improved. By this means, intensive human knowledge and labor can be spared. Existing works on this topic include Data Science Machine (DSM) [10], ExploreKit [9] and FeatureHub [22]. Besides, we have also seen commercial products such as FeatureLabs [10].

Thus, with such a rapid development of AutoML in both research and industry, we feel it necessary to summarize existing works and do a survey on this topic at this time. First, we define what is the AutoML problem. Then, we propose a general framework which summarize how existing approaches work towards AutoML. Such framework further motivates us to give taxonomies of existing works based on what (by problem setup) and how (by techniques) to automate. Specifically, problem setup helps us to clarify what learning tools we want to use, while techniques give us the technical methods and details to address the AutoML problem under the corresponding setup. Based on these taxonomies, we further give a guidance how AutoML approaches can be used and design. [1]

1.1 Contributions

Below, we summarize our contributions in this survey:

- We are the first to formally define the AutoML problem. The definition is general enough to include all existing AutoML problems, but also specific enough to clarify what is the goal of AutoML. Such definition is helpful for setting future research target in the AutoML area.
- We propose a general framework for existing AutoML approaches. This framework is not only helpful for setting up taxonomies of existing works, but also gives insights of the problems existing approaches want to solve. Such framework can act as guidance for developing new approaches.
- We systematically categorize the existing AutoML works based on "what" and "how". Problem setup is from "what" perspective, which justifies which learning tools we want to make automated. Techniques are from "how", they are methods to solve AutoML problems. For each category, we present detailed application scenarios as the reference.
- We provide a detailed analysis of approaches techniques. Compared to existing AutoML related surveys [29], we

1. In this survey we focus on the usage of existing techniques in AutoML, for individual reviews on related topics please refer to [23], [24], [25] for meta-learning, [26] for transfer learning, [27] for hyper-parameter optimization and [28] for neural architecture search.

not only investigate a more comprehensive set of existing works, but also present a summary of the insights behind each technique. This can serve as an good guideline not only for beginners' usage but also for future researches.
- We suggest four promising future research directions in the field of AutoML in terms of computational efficiency, problem settings, solution techniques and applications. For each direction, we provide a thorough analysis of its disadvantages in current work and propose future research directions.

1.2 Organization

The survey is organized as follows. The overview is in Section 2, which gives the definition of AutoML, the proposed framework of AutoML approaches, and taxonomies by problem setup and techniques of existing works. Section 3 describes the taxonomy by problem setup, and techniques are detailed in Section 4-6. Three application examples listed in Table 1 are detailed in Section 7. Finally, we end this survey with a brief history of AutoML, future works and a conclusion Section 8.

1.3 Notation

In the sequel, a machine learning tool is a method which can solve some learning problems in Figure 1, i.e., feature engineering, model selection and/or algorithm selection. We use term *configuration* to denote all factors but model parameters that influence the performance of a learning tool. Examples of configurations are, the hypothesis class of model, the features utilized by the model, the hyper-parameters that control the training procedure, and the architecture of network. Finally, we denote a machine learning tool as $F(\mathbf{x};\theta)$, where $\mathbf{x}$ is the model parameters learned by training and θ contains configurations of the learning tool.

2 Overview

In Section 1, we have shown why we need to do AutoML, which dues to both academic dream and industrial needs. In this section, we first define what is AutoML problem in Section 2.1. Then, in Section 2.2 we propose a framework of how AutoML problems can be solved in general. Finally, taxonomies of existing works based on "what to automate" and "how to automate" are presented in Section 2.3.

2.1 Problem Definition

Here, we define what is the AutoML problem, which is inspired by automation and machine learning. Based on the definition, we also explain core goals of AutoML.

2.1.1 AutoML from two Perspectives

From its name, we can see that AutoML is naturally the intersection of automation and machine learning. While automation has a long history, which can even date back to BC [30], machine learning was only invented decades ago [1]. The combination of these two areas has just become a hot research topic in recent years. The key ideas from these two fields and their impacts on AutoML are as follows.

- Machine learning, as in Definition 1, is specified by E, T and P, i.e., it tries to improve its performance on task T measured by P, when receiving training data E.

  Definition 1 (Machine learning [1]). *A computer program is said to learn from experience E with respect to some classes of task T and performance measure P if its performance can improve with E on T measured by P.*

  From this perspective, AutoML itself can also be seen as a very powerful machine learning algorithm that has good generalization performance (i.e., P) on the input data (i.e., E) and given tasks (i.e., T). However, traditional machine learning focuses more on inventing and analyzing learning tools, it does not care much about how easy can these tools be used. One such example is exactly the recent trend from simple to deep models, which can offer much better performance but also much hard to be configured [31]. In the contrast, AutoML emphasizes on how easy learning tools can be used. This idea is illustrated in Figure 2.

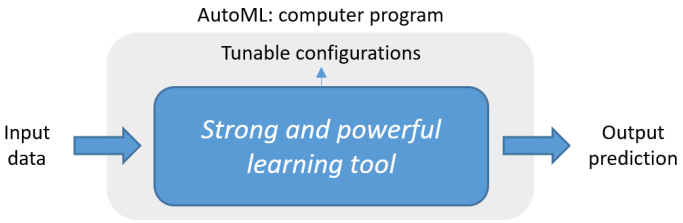


Fig. 2. AutoML from machine learning's perspectives.

- On the other hand, automation is the use of various control systems for operating underneath building blocks [32]. In pursuit of better predicting performance, configurations of machine learning tools should be adapted to the task with input data, which is often carried out manually. As shown in Figure 3, the goal of AutoML from this perspective is to construct high-level controlling approaches over underneath learning tools so that proper configurations can be found without human assistance.

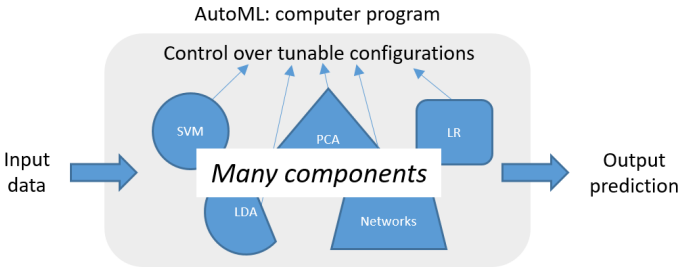


Fig. 3. AutoML from automation's perspectives.

These two perspectives are the main motivations for our AutoML's definition in the sequel.

2.1.2 The Definition of AutoML

From Section 2.1.1, we can see that AutoML not only wants to have good learning performance (from machine learning's perspective) but also requires such performance being achieved without human assistance (from automation's

TABLE 2
Why we need to have AutoML: an overview comparison of classical machine learning and AutoML.

	classical machine learning	AutoML
feature engineering	humans design, construct features from data	automation by the computer program
	humans process features making them more informative	
model selection	humans design or pick up some machine learning tools based on professional knowledge	
	humans adjust hyper-parameters of machine learning tools based on performance evaluation	
algorithm selection	humans pick up some optimization algorithms to find parameters	
computational budgets	not a main concern	execute within a limited computational budget
summary	human experts are involved in every aspect of machine learning applications	the program can be directly reused on other learning problems

perspective). Thus, an informal and intuitive description of AutoML can be expressed as

$$\max_{\text{configrations}} \text{ performance of learning tools}, \quad (1)$$
$$\text{s.t.} \begin{cases} \text{no human assistance} \\ \text{limited computational budget} \end{cases}.$$

Put it more formally, we describe what is AutoML in Definition 2. Such definition is inspired by Definition 1 and the fact that AutoML itself can also be seen as another machine learning approach (Figure 2).

Definition 2 (AutoML)**.** *AutoML attempts to construct machine learning programs (specified by E, T and P in Definition 1), without human assistance and within limited computational budgets.*

A comparison of classical machine learning and AutoML is in Table 2. Basically, in classical machine learning, human are heavily involved in the search for such learning tools, by operating feature engineering, model selection, hyper-parameter tuning, and network architecture design. As a result, human take the most labor and knowledge-intensive job in machine learning practices. However, in AutoML, all these can be done by computer programs. To understand the Definition 2 better, let us look back at those three examples in Table 1:

- *Automatic model selection*: Here, E denotes input training data, T is a classification task and P is the accuracy on the given task. When features are given, Auto-sklearn can choose proper classifiers and find corresponding hyper-parameters without human assistance.
- *Neural architecture search*: When we try to do some image classification problems with the help of NAS, E is the collection of images, T is the image classification problem, and P is the accuracy on testing images. NAS will automatically search for a neural architecture, i.e., a classifier based on neural networks, that has good performance on the given task.
- *Automatic feature engineering*: As the input features may not be informative enough, we may want to construct more features to enhance the learning performance. In this case, E is the raw feature, T is construction of features, and P is the performance of models which are learned with the constructed features. DSM [10] and ExploreKit [9] remove human assistance by automatically construct new features based on interaction among input features.

Finally, note that Definition 2 is general enough to cover most machine learning approaches that can be considered automatic. With this definition, a machine learning pipeline with fixed configurations, that do not adapt according to different E, T, and P, is also automatic. Approaches of this kind, though require no human assistance, are rather limited in their default performance and application scopes. Thus, they are not interesting, and will not be further pursuit in the sequel.

2.1.3 Goals of AutoML

Thus, from above discussion, we can see that while good learning performance is always desired, AutoML requires such performance can be obtained in a more special manner, i.e. without human assistance and within limited computational budgets. These set up three main goals for AutoML (Remark 2.1).

Remark 2.1 (Core goals)**.** *The three goals underneath AutoML:*

(A). Better Performance: good generalization performance across various input data and learning tasks;

(B). No Assistance from humans: configurations can be automatically done for machine learning tools; and

(C). Lower Computational budgets: the program can return an output within a limited budget.

Once above three goals can be realized, we can fast deploy machine learning solutions across organizations, quickly validate and benchmark the performance of deployed solutions, and let human focus more on problems that really need humans' engagements, i.e., problem definition, data collection and deployment in Figure 1. All these make machine learning easier to apply and more accessible for everyone.

2.2 Basic Framework

In Section 2.1, we have defined the AutoML problem (Definition 2) and introduced its core goals (Remark 2.1). In this section, we propose a basic framework for AutoML approaches.

2.2.1 Human Tuning Process

However, before that, let us look at how configurations are tuned by human. Such process is shown in Figure 4. Once a learning problem is defined, we need to find some

learning tools to solve it. These tools, which are placed in the right part of Figure 4, can target at different parts of the pipeline, i.e., feature, model or optimization in Figure 1. To obtain a good learning performance, we will try to set a configuration using our personal experience or intuition about the underneath data and tools. Then, based on the feedback about how the learning tools perform, we will adjust the configuration wishing the performance can be improved. Such a trial-and-error process terminates once a desired performance is achieved or the computational budgets are run out.

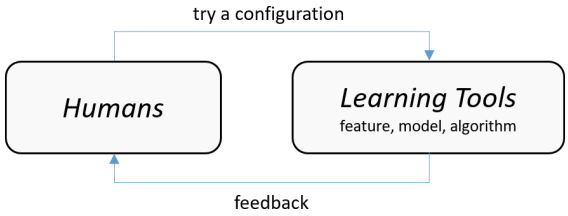


Fig. 4. The process of configurations tuned by humans.

2.2.2 Proposed AutoML Framework

Motivated by the human-involved process above and controlling with feedbacks in the automation [33], we propose a framework for AutoML, as shown in Figure 6. Compared with Figure 4, in this figure, an AutoML controller takes the place of human to find proper configurations for the learning tools. Basically, we have two key ingredients inside the controller, i.e., the optimizer and the evaluator. Their interactions with other components in Figure 6 are as follows:

- *Evaluator*: The duty of the evaluator is to *measure* the performance of the learning tools with configurations provided by the optimizer. After that, it generates *feedbacks* to the optimizer. Usually, to measure the performance of learning tools with the given configuration, the evaluator needs to train the model based on the input data, which can be time consuming. However, the evaluator can also directly estimate the performance based on external knowledge, which mimics humans' experience. Such estimation is very fast but may not be accurate. Thus, for the evaluator, it needs to be fast but also accurate in measuring the performance of learning tools.
- *Optimizer*: Then, for the optimizer, its duty is to update or generate the *configuration* for learning tools. The *search space* of the optimizer is defined by learning tools, and new configurations are expected to have better performance than previous ones. However, feedbacks offered by the evaluator are not necessary used by the optimizer. This depends on which type of the optimization algorithms we are utilizing. Finally, as the optimizer operates on the search space, we wish the search space can be easy and compact so that the optimizer can identify a good configuration with a few generated configurations.

In Table 10, we use examples from Table 1 to demonstrate how the proposed framework can cover existing works. Details of the examples are in Section 7. As we can see, such framework is general enough to cover nearly all existing works (e.g. [5], [7], [9], [10], [20], [34], [35], [36], [37], [38], [39], [40], [41]), but also precise enough to help us setup taxonomies for AutoML approaches in later Section 2.3. Furthermore, it offers future works to AutoML in Section 8.2.

2.3 Taxonomies of AutoML Approaches

In this section, we give taxonomies of existing AutoML approaches based on what and how to automate.

2.3.1 "What to automate": by Problem Setup

The choice of learning tools inspires the taxonomy based on problem setup in Figure 5(a), this defines "what" we want to make automated by AutoML.

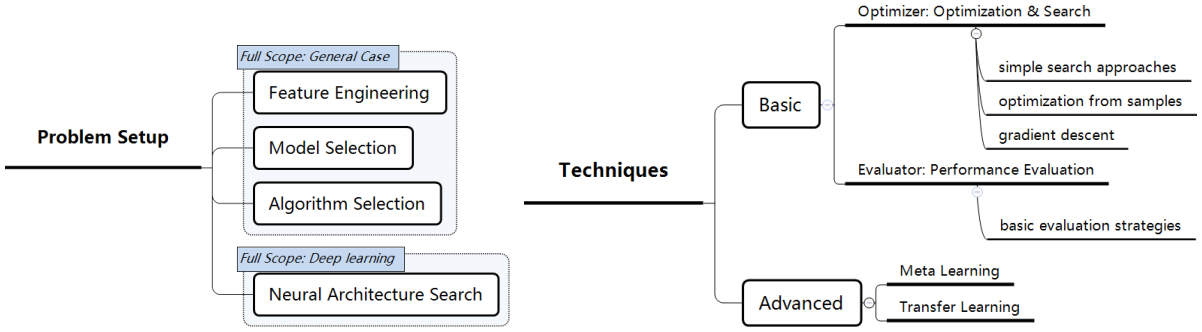
Basically, for general learning problems, we need to do feature engineering, model selection and algorithm selection. These three parts together make up the full scope of of general machine learning applications (Figure 1). We also list neural architecture search (NAS) there as a very important and special case. The reason is that NAS targets at deep models, where features, models and algorithms are configured simultaneously. The focus and challenges of AutoML problem under each setup are detailed in Section 3.

2.3.2 "How to automate": by Techniques

Figure 5(b) presents the taxonomy by AutoML techniques. These are the techniques used for the controller, and categorize "how" we solve an AutoML problem. In general, we divide existing techniques into basic and advanced ones:

- *Basic techniques*: As there are two ingredients, i.e., the optimizer and evaluator, in the controller, we categorize basic techniques based on which ingredient they operating on. The optimizer focus on the searching and optimizing configurations, and there are many methods can be used, from simple methods as grid search and random search [42] to very complex ones as reinforcement learning [7] and automatic differentiation [43]. However, for the evaluator, which mainly measures the performance of learning tools with current configurations by determine their parameters, there are not many methods can be taken as basic ones.
- *Advanced techniques*: The difference between basic and advance ones is that advance techniques cannot be used for searching configurations in Figure 5(b), they usually need to be combined with basic ones. Generally, there are two main methods fall into advanced techniques, i.e., meta-learning [24], [44] and transfer learning [26], they both try to make use of external knowledge to enhance basic ones for the optimizer and evaluator.

Note that, as E, T and P are also involved in the AutoML's definition (Definition 2), taxonomies of machine learning, e.g., supervised learning, semi-supervised learning and unsupervised learning, can also be applied for AutoML. However, these does not necessarily connect with removing human assistance in finding configurations (Figure 4). Thus, taxonomies here are done based on the proposed framework in Figure 6 instead. Finally, we focus on supervised AutoML approaches in this survey as all existing works for AutoML are supervised ones.



(a) "What to automate": by problem setup. (b) "How to automate": by techniques.

Fig. 5. AutoML approaches taxonomies by problem setup and techniques, which is inspired by the proposed framework in Figure 6. Taxonomy by problem setup depends on which learning tools we used, it clarifies "what" we want to make automated; taxonomy by techniques depends on the how we want to solve AutoML problems.

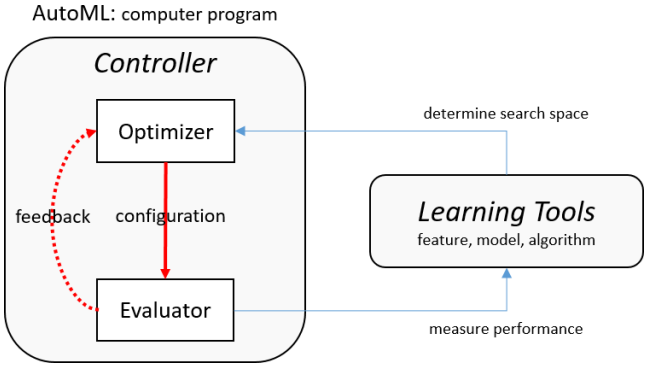


Fig. 6. Basic framework for how existing approaches solving AutoML problem. The dashed line (feedback) inside the controller, which depends on what techniques are used for the optimizer, is not a must.

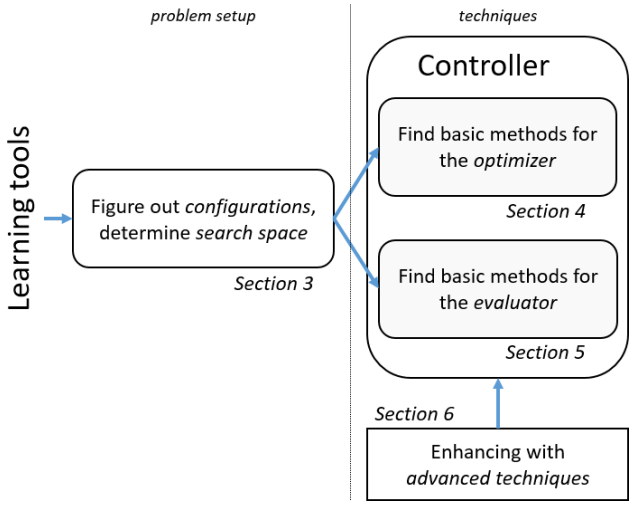


Fig. 7. Working flow of designing AutoML approaches based on the proposed framework (Figure 6) and taxonomies.

2.4 Working Flow based on Taxonomies

In the sequel, basic techniques and core issues they need to solve are introduced in Section 4 and 5 for the optimizer and evaluator respectively. After that advanced techniques are described in Section 6. The working flow of designing an AutoML approach is summarized in Figure 7, which also acts a a guidance through this survey.

3 Problem Settings

In this section, we give details on categorization based on problem setup (Figure 5(a)). Basically, it clarifies what to be automated. AutoML approaches need not solve the full machine learning pipeline in Figure 1, they can also focus on some parts of the learning process. Common questions need to be asked for each set up are:

Remark 3.1. *What learning tools can be designed and used? What are their corresponding configurations?*

By asking these questions we can then define the search space for AutoML approaches. An overview is in Table 3. In the sequel, we briefly summarize existing learning tools for each setup and what are the corresponding search space.

3.1 Feature Engineering

The quality of features, perhaps, is the most important perspective for the performance of subsequent learning models. Such importance is further verified by the success of deep learning models, which can directly learn a representation of features from the original data [63]. The problem of AutoML for feature engineering is to *automatically construct features from the data so that subsequent learning tools can have good performance*. The above goal can be further divided into two sub-problems, i.e., creating features from the data and enhance features' discriminative ability.

However, the first problem heavily depends on application scenarios and humans' expertise, there are no common or principled methods to create features from data. AutoML only makes limited progress in this direction, we take it as one future direction and discuss it in Section 8.2.2. Here, we focus on feature enhancing methods.

3.1.1 Feature Enhancing Tools

In many cases, the original features from the data may not be good enough, e.g., their dimension may be too high or samples may not be discriminable in the feature space. Thus, we may want to do some post-processing on these features. Fortunately, while human knowledge and assistance are still required, there are common methods and principle ways to enhance features. They are listed as follows:

TABLE 3
The taxonomy of existing AutoML approaches by problem setup. For each setup, we need to select or design some learning tools, and then figure out the resulting configurations (see Remark 3.1).

		target	configurations	examples
feature engineering		subsequent classifiers	feature sets	[9], [10], [45], [46], [47]
			selection of methods and their hyper-parameters	[5], [6], [34]
model selection		classifiers	selection of classifiers and their hyper-parameters	[5], [34], [37], [48], [49]
algorithm selection		optimization algorithms	selection of algorithms and their hyper-parameters	[38], [50], [51], [52], [53]
full scope	general	an union of above three aspects		[5], [6], [21], [34]
	NAS	neural networks	design of networks (e.g., network structure, learning rate)	[7], [20], [40], [54], [55], [56], [57], [58], [59], [60], [61], [62]

- *Dimension reduction*: It is the process of reducing the number of random variables under consideration by obtaining a set of principal variables, which is useful when there exists much redundancy among features or the feature dimension is too high. It can be divided into feature selection and feature projection. Feature selection tries to select a subset of features from the original ones, popular methods are greed search and lasso. Feature projection transforms original features to a new space, of which the dimension is much smaller, e.g., PCA [64], LDA [65] and recently developed auto-encoders [66].
- *Feature generation*: As original features are designed from humans, there are usually unexplored interactions among them which can significantly improve learning performance. Feature generation is to construct new features from the original ones based on some pre-defined operations [ref], e.g., multiplication of two features and standard normalization. It is usually modeled as a searching problem in the space spanned by operations on original features. Many search algorithms has been applied, e.g., greedy search [9] and evolution algorithms [45], [67].
- *Feature coding*: The last category is feature coding, which re-interprets original features based on some dictionaries learned from the data. After the coding, samples are usually lifted in another feature space, which is much higher than the original one. Since the dictionary can capture the collaborative representation in the training data, samples are not discriminable in the original space become separable in the new space. Popular examples are sparse coding [68] (and its convolutional variants [69]) and local-linear coding [70]. Besides, kernel methods can also be seen as feature coding, where basis functions are the dictionary. However, kernel methods have to be used with SVM and basis functions are designed by hand not driven by data.

All above tools are not automatic. While there are practical suggestions for using above feature enhancing tools, when facing with a new task, we still need to try and test.

3.1.2 Search Space

There are two types of search space for above feature enhancing tools. The first one is made up by hyper-parameters of these tools, and configuration exactly refers to these hyper-parameters. It covers dimension reduction and feature coding methods, (e.g., [5], [6], [34]). For example, we need to determine the dimension of features after PCA, and the level of sparsity for sparse coding. The second type of search space comes from feature generation, (e.g., [9], [10], [45], [46], [47]). The space is spanned by the combination of predefined operations with original features. One example of new feature generated from plus, minus and times operation is shown in Figure 8. For these methods, a configuration is a choice of features in the search space.

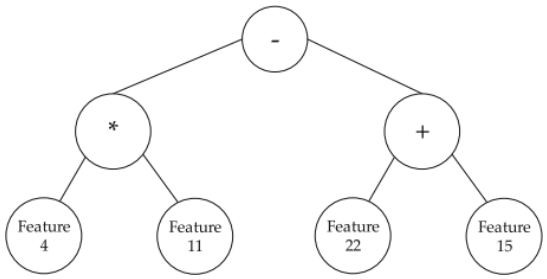


Fig. 8. An example of generated feature, which is based on times, minus and plus operation from the 4th, 11st, 22nd and 15th original features (the image is from [45]).

3.2 Model Selection

Once features have been obtained, we need to find a model to predict labels. Models selection contains two components, i.e., picking up some classifiers and setting their corresponding hyper-parameters. The problem of AutoML here is to *automatically select classifiers and setup their hyper-parameters so that good learning performance can be obtained*.

3.2.1 Classification Tools

Many classification tools have been proposed in the literature, e.g., tree classifiers, linear classifiers, kernel machines and, more recently, deep networks. Each classifier has its own strength and weakness in modeling underneath data. For example, tree classifiers generally outperform linear ones. However, when the feature dimension becomes high, it becomes extremely expensive and difficulty to train tree classifier. In this case, linear classifiers are preferred. Some out-of-box classifiers implemented in scikit-learn are listed in Table 4. Traditionally, the choice among different classifiers are usually made by human from his experience in a trial-and-error manner.

As can be seen from Table 4, we also have hyper-parameters associated with each classifier. They are usually determined by grid-search, which is also the standard practice in machine learning communities. However, the size of search grids grows exponentially with the number of

TABLE 4
Example classifiers in Scikit-Learn and their hyper-parameters. Generally, hyper-parameters can be (a) discrete, e.g., number of neighbors in kNN, or (b) continuous. e.g., the value of penalty in logistic regression.

	number of hyper-parameters		
	total	discrete	continuous
AdaBoost	4	1	3
Bernoulli naive Bayes	2	1	1
decision tree	4	1	3
gradient boosting	6	0	6
kNN	3	2	1
linear SVM	4	2	2
kernel SVM	7	2	5
random forest	5	2	3
logistic regression	10	4	6

hyper-parameters. Thus, grid-search is not a good choices for complex models. In this case, again, the importance of hyper-parameters are first evaluated by humans, and then those not important or insensitive ones are pre-defined. Such process still needs human assistance, and may lead to sub-optimal performance as the possible settings of hyper-parameters are not sufficiently explored.

3.2.2 Search Space

From above, we can see that configurations under the context of model selection are choices of classifiers and their hyper-parameters (e.g., [5], [34], [37], [48], [49]). These two parts make up the search space here. Choices of classifiers can simply be modeling as a discrete variable with 1 indicates using the classifier and 0 stands fro not using. Properties of hyper-parameters depends on the design and implementation of models, i.e., the number of nearest neighbor is discrete while the penalty parameter for logistic regression is continuous.

3.3 Algorithm Selection

The last and the most time consuming step of machine learning is to find parameters of learning tools, where optimization tools are usually involved. Transitionally, as leaning tools are usually very simple, optimization is not a concern, the performance obtained from various optimization tools are nearly the same. Efficiency is the main focus on the choice of optimization tools. However, as the learning tools get increasing more complex. Optimization is not only the main consumer of computational budgets but also has a great impact on the performance of learning as well. Thus, the goal of AutoML here is to *automatically find an optimization tools so that both efficiency and performance can be balanced*.

3.3.1 Optimization Tools

For each learning tool, many algorithms can be used. Some popularly tools for minimizing smooth objective functions, like logistic regression, are summarized in Table 5. While GD do not involve with any extra-parameters, it suffers from slow convergence and expensive per-iteration complexity. L-BFGS is more expensive but converges faster, and each iteration is very cheap in SGD but many iterations are needed before convergence.

TABLE 5
Some popular optimization tools for minimizing smooth objectives. L-BFGS needs to select the length of stored gradient (discrete); SGD needs to determine mini-batch size (discrete) and step-size (e.g. $\eta_0/(1+\lambda\eta_0 t)^c$ where t is the number of iterations, η_0, λ and c are continuous hyper-parameters [71]).

	number of hyper-parameters		
	total	discrete	continuous
gradient descent (GD)	0	0	0
Limited memory-BFGS (L-BFGS)	1	1	0
stochastic gradient descent (SGD)	4	1	3

3.3.2 Search Space

Traditionally, both the choices of optimization tools and their hyper-parameters are made by humans. These again based on humans' understanding of learning tools and observations of the training data. The search space is determined by configurations of optimization tools, which contains the choice of optimization tools and the values of their hyper-parameters (e.g, [38], [50], [51], [52], [53]).

3.4 Full Scope

In the last section, we discuss the full pipeline in Figure 1. We divide it into two cases here. The first one is general case, learning tools for this case are just the union of previous ones discussed in Section 3.1-3.3. The resulting searching space is also a combination of previous ones. However, the search can be done in two manners, either by reusing methods for each setup separately and then combining them together, or directly search through the space spanned by all configurations. Note that, there can be some hierarchical structure in the space (e.g., [5], [6], [21], [34]), as choices of optimization tools depends on which classifier is used.

The second one is network architecture search (NAS), which targets at searching good architectures for deep networks (e.g., [7], [20], [40], [54], [55], [56], [57], [58], [59], [60], [61], [62], [72]). There are two main reasons why we put is here in parallel to the full scope. First, NAS itself is an extremely hot research topic now where many papers have been published. The second reason is that the deep networks are very strong learning tools, which can learn directly from data, and SGD is the main choice for optimization.

3.4.1 Network Architecture Search

Before describing the search space of NAS, let us look at what is a typical CNN architecture. As in Figure 9, basically, CNN is mainly made up by two parts, i.e., a series of convolutional layers and a fully connected layer in the last.

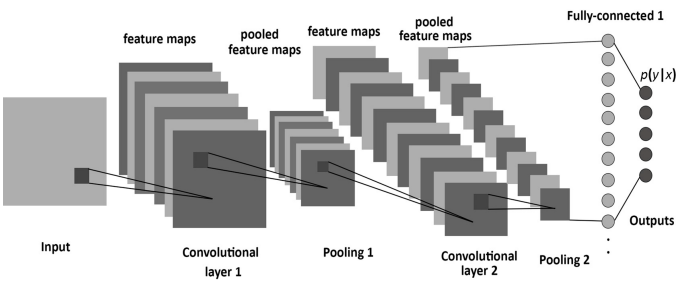


Fig. 9. A very typical CNN architecture, which contains filters, pooling and jump connections (the image is from [73]).

The performance of CNN is mostly influenced by the design of convolutional layers [74]. Within one layer, some common design choices are listed in Figure 10. More importantly, comparing with model selection in Section 3.2, the model complexities of deep network can also be taken into search, i.e., we can add one more layer or jump the connection between layers. Such key difference is the motivation to use reinforcement learning [19] in NAS. Thus, the search space is made up by above design choices and the hyper-parameters in SGD. One configuration here for NAS is one point in such a search space.

- 3x3 depthwise-separable convolution
- 5x5 depthwise-separable convolution
- 7x7 depthwise-separable convolution
- 1x7 followed by 7x1 convolution
- identity
- 3x3 average pooling
- 3x3 max pooling
- 3x3 dilated convolution

Fig. 10. Some common design choices for one convolutional layer in a CNN (the image is from [8]).

In this survey, we focus on CNN, ideas presented here can be similarly applied for other deep architectures, such as long-short-term-memory (LSTM) [75] and deep sparse networks [76].

4 Basic Techniques for Optimizer

Once the search space is defined, as in the proposed framework (Figure 6), we need to find an optimizer to guide the search in the space. In this section, we discuss the basic techniques for the optimizer.

Remark 4.1. *Three important questions here are*

(A). what kind of search space can the optimizer operate on?

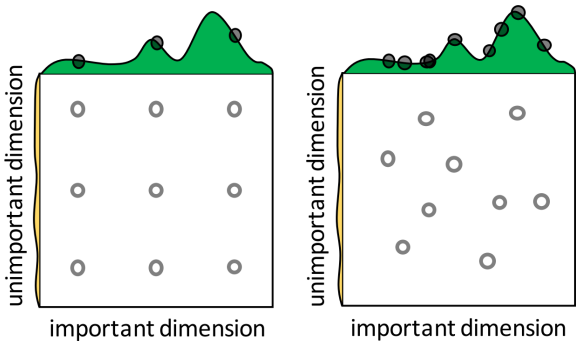
(B). what kind of feedbacks it needs?

(C). how many configurations it needs to generate/update before a good one can be found?

The first two questions determine which type of techniques can be used for the optimizer, and the last one clarifies the efficiency of techniques. While efficiency is also a big pursue in AutoML (see Remark 2.1), we do not categorizes existing techniques here based on it. This is because the search space is so complex where convergence rates for each technique are hard to analyze and advanced techniques (Section 6) can accelerate basic ones in various ways. Thus, in the sequel, we divide those techniques into three categories, i.e., simple search approaches, optimization from samples, and gradient descent, based on the first two questions. An overview of the comparison among these techniques are in Table 6.

4.1 Simple Search Approaches

Simple search is a kind of naive search approach, they make no assumptions about search space and they do not need any feedbacks from the evaluator. Each configuration in the search space can be evaluated independently. Simple search approaches such as grid search, random search are widely used in configuration optimization. Grid search (brute-force) is the most traditional way of finding hyper-parameters. To get the optimal hyper-parameter setting, grid search have to enumerate every possible configurations in search space. Thus, discretization is necessary when search space is continuous. Random search [42] can better explore the search space as more positions will be evaluated (Figure 11). However, they both suffer from the curse of dimensionality while the dimensionality is increasing.



(a) Grid search. (b) Random search.

Fig. 11. Illustration of grid and random search with 9 trials in 2-D search problem. This figure also illustrates that random search does more exploration than grid search when the number of trials is same (the image is from [42]).

4.2 Optimization from Samples

Optimization from samples [77] is a kind of smarter search approach compared with simple ones in Section 4.1. It iteratively generates new configurations based on previous ones. Thus, it is also generally more efficient than simple search methods. Besides, it does not make specific assumptions on the objective as well.

In the sequel, according to different optimization strategies, we divide existing approaches into three categories, i.e., heuristic search, model-based derivative-free optimization and reinforcement learning.

4.2.1 Heuristic Search

Heuristic search methods are often inspired by biologic behaviors and phenomenons. It is widely used on optimization problems, which are nonconvex, nonsmooth or even noncontinuous. They are all population-based optimization methods, and different with each other in how they generate and select populations. Some popular heuristic search methods are listed as follow:

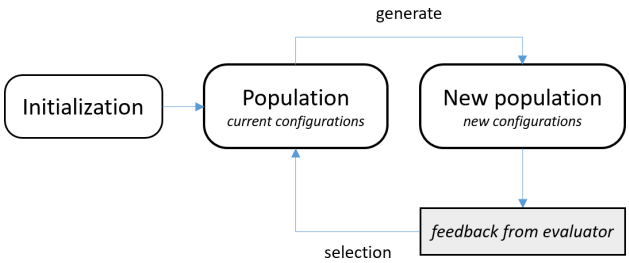


Fig. 12. Work flow of heuristic search. It is the population-based search approach, and starts with a initialization process.

- *Particle swarm optimization* (PSO) [48]: PSO is inspired by social behavior of bird flocking or fish schooling. It optimizes by searching the local area around the best

TABLE 6
Comparison of various techniques for the optimizer based on Remark 4.1.

type	method	continuous	discrete	examples	feedback in examples
simple search	random, grid search	√	√	[42]	none
optimization from samples	evolutionary algorithm	√	√	[12]	accuracy on validation set
	Bayesian optimization	√	√	[34]	accuracy on validation set
	reinforcement learning	√	√	[20]	accuracy on validation set (reward) and a sequence of configurations (state)
gradient descent	reversible	√	×	[36]	accuracy on validation set and gradients w.r.t hyper-parameters

sample. PSO has few hyper-parameters itself and is easy to be parallelized.

- *Evolutionary algorithms* [78]: Evolutionary algorithms are inspired by biological evolution. Mutation and selection are main components. Some of state-of-the-art methods such as CMA-ES [79] are applied to solve kinds of sophisticated optimization problems.

The above methods have been applied in AutoML (e.g., [45], [48], [80], [81], [82], [83], [84]), and they all follow framework in Figure 12. Basically, a bunch of configurations (population) are maintained. In each iteration, first, new configurations are generated by crossover or mutation, then these configurations are measured using feedbacks from the evaluator and only a few are kept for the next iteration.

4.2.2 Model-Based Derivative-Free Optimization

Model-based derivative-free optimization (Figure 13) builds a model based on visited samples, which helps to generate more promising new samples. The popular methods are Bayesian optimization, classification-based optimization and optimistic optimization:

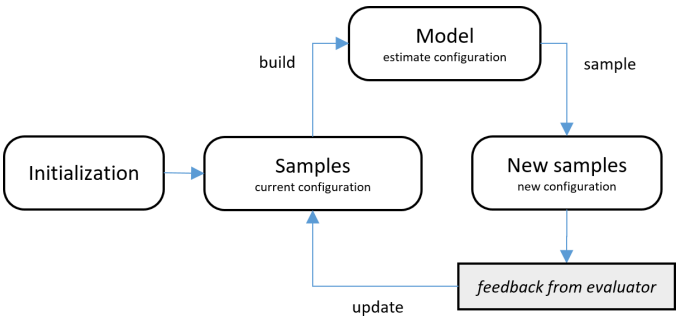


Fig. 13. Work flow of model-based derivative-free optimization. It is different from the heuristic search. The most important component of model-based optimization is the model process on previous samples.

- *Bayesian optimization* [85], [86]: Bayesian optimization builds a probabilistic surrogate function on search space by Gaussian process or other model (e.g., decision tree [87], random forest [88]). Then, it chooses the next sample by optimizing the acquisition function which is based on the surrogate function. Because of the excellent performance on expensive optimization, Bayesian optimization is popularly used in AutoML.
- *Classification-based optimization* [39], [89]: Based on previous samples, classification-based optimization models search space by learning a classifier. Through the classifier, the search space is divided into positive and negative areas. Then, new samples are generated on the positive area, from which, it is more likely to gets better samples. The model method is simple and effective. Thus, classification-based optimization has the high-efficiency and good scalability.
- *Simultaneous Optimistic optimization* (SOO) [90]: SOO applies a tree structure to balance exploration and exploitation on search space. SOO can get global optimum when the objective function is local-Lipschitz continuous. But it also suffers the curse of dimensionality because the tree grows extremely hard when dimensionality of objective function is high.

When used in AutoML, the above methods all share the same framework as in Figure 13. Compared with Figure 12, the main difference is that new configurations (i.e., samples) are generated from the model instead of fixed rules.

4.2.3 Reinforcement Learning

Reinforcement learning (RL) [19] is a very general and strong optimization framework, which can solve problems with delayed feedbacks. Its general framework when used in AutoML (Figure 14). Basically, the policy in RL acts as the generator, and its actual performance in the environment is measured by the evaluator. However, unlike previous methods, the feedbacks (i.e., reward and state) do not need to be immediately returned once an action is taken. They can be returned after performing a sequence of actions.

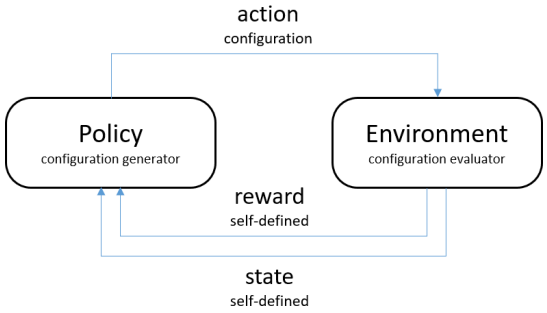


Fig. 14. Framework of reinforcement learning in AutoML. The action that policy generates is often the model configuration. But the reward and the state should be designed according to AutoML problems.

Due to above unique property, RL is popularly used in NAS (i.e., [7], [13], [20], [21], [91], [92]). The reason is that CNN can be built layer-by-layer, and the design of one layer can be seen as one action given by the generator. Thus, the iterative architecture generation naturally follows the property of RL (see details in Section 7.2). However, again due to delayed feedbacks, AutoML with reinforcement learning is high source-consuming. More efficient methods needs to be explored.

4.3 Gradient descent

Because optimization problems of AutoML is very complex, which is perhaps not differentiable or even not continuous. Thus, gradient descent is hard to be an effective optimizer. However, focusing on some differentiable loss function [93], e.g., squared loss and logistic loss, continuous hyper-parameters can be optimized by gradient descent. Compared with above methods, gradients offer the most accurate information where better configurations locates.

For these type of methods, unlike traditional optimization problem whose gradients can be explicitly derived from the objective, the gradients need to be implicitly computed here. Usually, this can be done by with finite differentiation [93]. Another way of computing the exact gradients is through reversible learning [36], [43], [94]. It has be applied into deep learning hyper-parameter search. For traditional machine learning, the approximate gradient was proposed to search continuous hyper-parameters [95]. Through this inexact gradient, hyper-parameters can be updated before the model parameters have converged.

4.4 Others

Finally, in this section, we briefly talk about methods for the structured search space or which can change the landscape of the search space. Such techniques are used and developed case-by-case. For example, greedy search is used in [9] for searching space spanned by feature combinations (Section 3.1.2), which is mainly motivated by the prohibitive large searching space. Then, in NAS, as some design choices are discrete which are not differential, soft-max are used in [41] to change the search space to a differential one, which enables the usage of gradient descent instead of RL.

5 Basic Techniques for Evaluator

Previously, in Section 4, we have discussed how to choose a proper basic technique for the optimizer. In this section, we talk about techniques for another component, i.e., the evaluator in Figure 6. For learning tools, once their configurations are updated, the evaluator needs to measure the corresponding performance on the validation set. Such process is usually very time consuming as it involves with parameter training.

Remark 5.1. *There important questions here are:*

(A). Can the technique provide fast evaluation?

(B). Can the technique provide accurate evaluation?

(C). What kind of feedback needs to be offered by the evaluator?

There is a trade off between the first two questions, fast evaluation usually leads to worse evaluation, i.e., less accurate with larger variance. This is illustrated in Figure 15. Thus, for evaluator's techniques, they wish to lie in the left top of Figure 15 with smaller variance.

The last question in Remark 5.1 is a design choice, it has to be combined with choices of the optimizer. For example, as in Table 6, grid search and random search need not give feedback to the optimizer, each configuration is run independently. However, for gradient descent methods, we not only needs to report the obtained performance on the validation set, but the gradient w.r.t. the current configuration has to be computed as well.

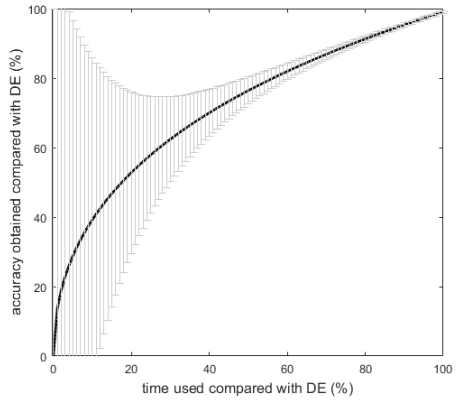


Fig. 15. The trade off between evaluation's accuracy and time, where DE denotes direct evaluation (see Section 5.1) and both time and accuracy are measured relatively to that of DE. The gray lines indicate variance in accuracy obtained.

5.1 Techniques

Unlike basic techniques for the optimizer, there are not many techniques can be used as basic ones for the evaluator. We list them as follow:

- *Direct evaluation*: This is the simplest method, where the parameters are obtain by directly training the parameters on the training set, and then the performance is measured on the validation set. It is the most expensive method, but also offers the most accuracy evaluation;
- *Sub-sampling*: As the training time depends heavily on the amount of training data, an intuitive method to make evaluation faster is to train parameters with a subset of the training data. This can be done by either using a subset of samples or a subset of features;
- *Early stop*: For some bad configurations, even with enough training time, their performance will not eventually gets much better. Empirically, such configurations usually can be easily identified by their performance on the validation set at the very beginning of the training [60], [88], [96]. In this case, we can early terminate the training, and let the evaluator offers a bad feedback;
- *Parameter reusing*: Another technique is to use parameters, that are obtained from evaluation of previous configurations, to warm-start the parameters need to be trained for the current evaluation. Intuitively, parameters for similar configurations can be close with each other. Thus, this technique can be useful when changes in the configuration between previous one and current one are not big.
- *Surrogate evaluator*: For configurations that can be readily quantized, one straightforward method to cut down the evaluation cost is to build a model that predicts the performance of given configurations, with experience of past evaluations [38], [54], [96], [97]. These models, serving as surrogate evaluators, spare the computationally expensive model training, and significantly accelerate AutoML. Surrogate evaluator is also an accelerating approach that trades evaluation accuracy for efficiency, it can be used to predict running time, parameters and predicting per-

formance of learning tools. The use of surrogate models is comparatively limited for configurations other than hyperparameters since they are hard to quantize. In Section 6.1, we will introduce meta-learning techniques that are promising to address this problem.

As a summary, due to simplicity and reliability, *Direct evaluation* is perhaps the most commonly used basic technique for the evaluator. Sub-sampling, early stop, and parameter reusing enhance *Direct evaluation* in various directions, and they can be combined for faster and more accurate evaluation. However, the impact of these three techniques depends on AutoML problems and data, it is hard to conclude to which content they can improve upon *Direct evaluation*. Note that, while surrogate is also used in Section 4.2.2 (i.e., "Model" in Figure 13), it is used to generate new configurations which are more likely to have good performance. The surrogate evaluator here will determine the real performance of the given configuration, and offer feedbacks for the optimizer for subsequent updates. Finally, while basic techniques are few here, various ones can be designed based on, e.g., transfer learning and meta-learning. They will be discussed in Section 6.

6 Advanced Techniques

In previous sections, we discussed the general framework to automatically construct learning tools for given learning problems. The framework features a search procedure that comprises configuration generation and evaluation. In this section, we review advanced techniques that can improve the efficiency and performance of AutoML, by putting them into our proposed framework. Two major topics of this section are: 1) meta-learning, where meta-knowledge about learning is extracted and meta-learner is trained to guide learning; 2) transfer learning, where transferable knowledge is brought from past experiences to help upcoming learning practices.

6.1 Meta-Learning

Though with various definitions, meta-learning in general learns how specific learning tools perform on given problem from past experiences, with the aim to recommend or construct promising learning tools for upcoming problems. Meta-learning is closely related to AutoML since they share same objectives of study, namely the learning tools and learning problem. In this section, we will first briefly introduce the general framework of meta-learning and explain why and how meta-learning can help AutoML. Then, we review existing meta-learning techniques by categorizing them into three general classes based on their applications in AutoML: 1) meta-learning for configuration evaluator; 2) meta-learning for configuration optimizer; and 3) meta-learning for dynamic configuration adaptation.

6.1.1 General Meta-learning Framework

Meta-learning satisfies the definition of machine learning (Definition 1). It is, however, significantly different from classical machine learning since it aims at totally different tasks and, consequently, learns from different experiences. Table 8 provides an analogy between meta-learning and classical machine learning, indicating both their similarities and differences.

Like classical machine learning, meta-learning is achieved by extracting knowledge from experience, training learners based on the knowledge, and applying the learners on upcoming problems. Figure 16 illustrates the general framework of meta-learning. First, learning problems and tools are characterized. Such characteristics (e.g., statistical properties of the dataset, hyperparameters of learning tools) are often named meta-features, as thoroughly reviewed in [23], [24], [98]. Then, meta-knowledge is extracted from past experiences. In addition to meta-features, empirical knowledge about the goal of meta-learning, such as performance of learning tools and the promising tools for specific problems, is also required. Afterwards, meta-learners are trained with the meta-knowledge. Most existing machine learning techniques, as well as simple statistical methods, can serve to generate the meta-learners. The trained meta-learner can be applied on upcoming, characterized learning problems to make predictions of interest.

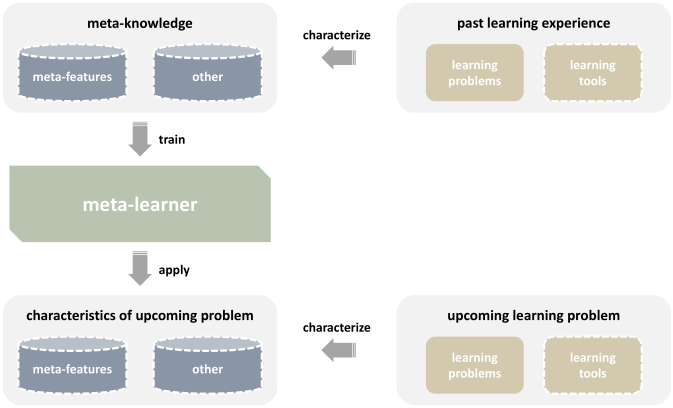


Fig. 16. The general framework of meta-learning techniques in AutoML.

Meta-learning helps AutoML, on the one hand, by characterizing learning problems and tools. Such characteristics can reveal important information about the problems and tools, for example, whether there are concept drift in the data, or whether a model is compatible for particular machine learning tasks. Furthermore, with these characteristic, similarities among different tasks and tools can be evaluated, which enables knowledge reuse and transfer between different problems. A simple but widely-used approach is to recommend configuration for a new task using the empirically best configuration in a neighborhood of this task in the meta-feature space. On the other hand, the meta-learner encodes past experience and acts as a guidance to solve future problems. Once trained, the meta-learners can fast evaluate configurations of learning tools, sparing the computational expensive training and evaluation of models. They can also generate promising configurations, which can directly specify a learning tool or serve as good initialization of the search, or suggest effective search strategies. Hence, meta-learning can greatly improve the efficiency of AutoML approaches.

In order to apply meta-learning in AutoML, we need to figure out the purpose of meta-learning, and the corresponding meta-knowledge and meta-learners, as noted

TABLE 7
The required meta-knowledge and meta-learners for different purposes (in existing literature).

applications		meta-knowledge (extracted from past experience)			meta-learner (applied on upcoming problems)	
		learning problem	learning tool	other	input	output
configuration evaluation (evaluator)	model evaluation	meta-features of data	meta-features of models (optional)	performance, or applicability, or ranking of models	meta-features of data and models	performance, or applicability, or ranking of models
	general configuration evaluation	meta-features of data (optional)	configurations, or meta-features of configurations (optional)	performance of configurations	meta-features of data, and configurations or meta-features of configurations	performance of configurations
configuration generation (optimizer)	promising configuration generation	meta-features of data	-	well-performing configurations	meta-features of data	promising configurations
	warm-starting configuration generation	meta-features of data	-	well-performing configurations	meta-features of data	promising initial configurations
	search space refining	-	configurations	importance of configurations, or promising search regions	configurations of learning tools	refined search space
for dynamic configuration adaptation	concept drift detection	statistics of data, or attributes	-	indicator (whether concept drift presented, optional)	statistics of data, or attributes	indicator, or indicating attributes
	dynamic configuration adaptation	meta-features of data	-	well-performing configurations	meta-features of current data	promising configuration

TABLE 8
Comparison between meta-learning and classical machine learning.

	classical machine learning	meta-learning
tasks	to learn and use knowledge about instances	to learn and use knowledge about learning problems and tools
experiences	about instances	about learning problems and tools
method	to train learners with experiences and apply them on future tasks	

in Remark 6.1. Table 7 summarizes the meta-knowledge and meta-learners that should be extracted and trained for different purposes, according to existing works in the literature.

Remark 6.1. *To apply meta-learning in AutoML, we should determine:*

(A). what is the purpose to apply meta-learning;

(B). what meta-knowledge should be extracted to achieve the purpose;

(C). what meta-learners should be trained to achieve the purpose.

6.1.2 Configuration Evaluation (evaluator)

The most computation-intensive step in AutoML is configuration evaluation, due to the cost of model training and validation. Meta-learners can be trained as surrogate evaluators to predict performances, applicabilities, or ranking of configurations. We summarize representative applications of meta-learning in configuration evaluation as follow:

- *Model evaluation*: the task is to predict, given a learning problem, often specified by the data set, whether or how a learning algorithm is applicable, or a ranking of candidate algorithms, so that the most suitable and promising algorithm can be selected. The meta-knowledge includes the meta-features of learning problems and the empirical performance of different models, and optionally the meta-features of models. The meta-learner is trained to map the meta-features to the performance [50], [99], [100], applicability [101], [102], or ranking [103], [104], [105], [106] of models. More recent research on this topic include active testing [**?**], [107], runtime prediction [52], [108], and more sophiscated measurements for models [109], [110]. A more complete review of on this topic can be found in [98].
- *General configuration evaluation*: the evaluation for other kinds of configurations can equip meta-learning in similar ways: in ExploreKit [9], ranking classifiers are trained to rank candidate features; in [111], meta-regressor is trained to score kernel widths as hyperparameters for support vector machines.

In short, with the purpose to accelerate configuration evaluation, meta-learners are trained to predict the performance or suitability of configurations. When used in the configuration generation procedure, such meta-learners can significantly cut down the number of actual model training. Furthermore, in the configuration selection setting, where all possible choices have been enumerated, best configurations can be directly selected according to the scores and rankings predicted by the meta-learner.

6.1.3 Configuration Generation (optimizer)

Meta-learning can also facilitate configuration generation by learning, e.g., configurations for specific learning tasks, strategies to generate or select configurations, or refined search spaces. These approaches, in general, can improve the efficiency of AutoML:

- *Promising configuration generation*: the purpose is to directly generate well-performing configurations for given learning problem. For this purpose, meta-knowledge indicating the empirically good configurations are extracted, and the meta-learner take the characteristics of learning problem as input and predict promising configurations, such as kernel [112], adaptive network architectures [113], [114].
- *Warm-starting configuration generation*: meta-knowledge utilized in *promising configuration generation* can also be exploited to better initialize configuration search. The basic approach is, given a new learning task, to find the past tasks that are closest to it in the meta-feature space, and use their best performing configurations to initialize search. Most work of this kind focus on hyperparameter tuning, with particle swarm optimization [115], [116], evolutionary algorithm [117], and sequential model-based optimization [5], [118], [119].
- *Search space refining*: meta-learning can accelerate configration search by refining the search space. Existing works of this line make effort to evaluate the importance of configurations [120], [121], or identify promising regions in the search space [122].

6.1.4 Dynamic Configuration Adaptation

So far we have focused on the difference among different learning problems and tools, which raises the need of AutoML. However, in the real life, the data distribution varies even in a single data set, especially in data streams. Such change in data distribution is often termed as "concept drift". In classical machine learning practices, concept drift is often priorly assumed or posteriorly detected, followed by specific design so that the learning tool can adapt to such drift. Meta-learning can help to automatic this procedure by detecting concept drift and dynamically adapt learning tools to it:

- *Concept drift detection*: with statistics of data or attributes, we can detect if concept drift present in a learning problem. In [123], attributes that might provide contextual clues, which indicate the changes in concept, are identified based on meta-learning. In [124], a non-parametric approach is proposed to detect concept drift. A new class of distance measures are designed to indicate changes in data distribution, and concept drift is detected by monitoring the changes of distribution in a data stream.
- *Dynamic configuration adaptation*: once the concept drift is detected, configuration adaptation can be carried out by predicting the promising configurations for current part of data [125], [126], [127]. Such approaches are similar to those in *promising configuration generation*.

Summary. We have so far reviewed major meta-learning techniques in the context of AutoML, however, applying meta-knowledge requires certain efforts, as will be discussed in Section 8.2.3.

6.2 Transfer Learning

Transfer learning, according to the definition in [26], tries to improve the learning on target domain and learning task, by using the knowledge from the source domain and learning task. In the context of AutoML, the source and target of transfer are either configuration generations or configuration evaluations, where the former setting transfers knowledge among AutoML practices and the latter transfers knowledge inside an AutoML practice. On the other hand, transferable knowledge that has been hitherto exploited in AutoML includes but is not limited to: 1) learned models or their parameters; 2) configurations of learning tools; 3) strategies to search for promising learning tools. Figure 17 illustrates how transfer learning works in AutoML. Remark 6.2 points out the key issues in applying transfer learning, and Table 6.2 summarized the different source, target, and transferable knowledge involved in transfer learning in the existing AutoML literature.

Remark 6.2. *To apply transfer learning in AutoML, we need to determine:*

(A). what is the purpose of knowledge transfer;
(B). what are the source and target of knowledge transfer;
(C). what knowledge to be transfered.

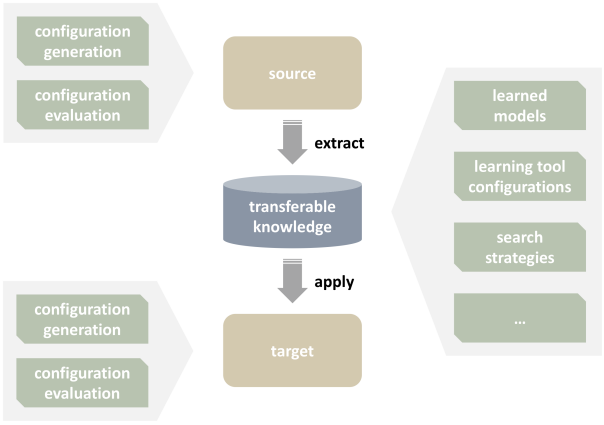


Fig. 17. An illustration of transfer learning in AutoML.

In the remaining of this section, we will review the transfer learning techniques that have been employed to help: 1) configuration generation (targeted at optimizer), and 2) configuration evaluation (targeted at evaluator).

6.2.1 Configuration Generation (optimizer)

In AutoML, the search for good configurations is often computational expensive due to the costly evaluations and extensive search spaces. Transfer learning has been exploited to reuse trained surrogate models or promising search strategies from past AutoML search (source) and improve the efficiency in current AutoML task (target):

- *Surrogate model transfer*: sequential model-based optimization (SMBO) for hyperparameters suffers from the cold-start problem, as it is expensive to initialize the surrogate model from scratch for every AutoML problem. Transfer learning techniques are hence proposed to reuse the knowledge gained from past experiences,

TABLE 9
Different source, target and knowledge for transfer learning in AutoML (in existing literature).

applications		source	target	knowledge
configuration generation (optimizer)	surrogate model transfer	past hyperparameter optimization	current hyperparameter optimization	surrogate model, or model components
	network block transfer	past network architecture search	current network architecture search	network building blocks
configuration evaluation (evaluator)	model parameter transfer	past architecture evaluation	current architecture evaluation	model parameter
	function-preserving transformation	past architecture evaluation	current architecture evaluation	the function represented by the network

by transferring the surrogate model [128] or its components such as kernel function [129].

- *Network block transfer*: transfer learning is especially widely-used in the realm of network architecture search due to the transferability of networks. In [21], [59], the NAS problem is converted to searching for architecture building blocks. Such blocks can be learned with low costs on small data sets and transferred to larger ones.

It should be noted that multi-task learning, a topic closely related to transfer learning, is also employed to help configuration generation. In [130], Bayesian optimization is accompanied with multi-task gaussian process models so that knowledge gained from past tuning tasks can be transfered to warm-start search. In [131], a multi-task neural AutoML controller is trained to learn hyperparameters for neural networks.

6.2.2 Configuration Evaluation (evaluator)

In the search for promising learning tools, a great number of candidate configurations need to be evaluated. In common approaches, such evaluation involves expensive model training. By transferring knowledge from previous configuration evaluations, we can avoid training model from scratch for the upcoming evaluations and significantly improve the efficiency. Based on the well-recognized and proven transferability of neural networks, transfer learning techniques have been widely employed in NAS approaches to accelerate the evaluation of candidate architectures:

- *Model parameter tranfer*: the most straightforward method is to transfer parameters from trained architectures to initialize new ones. According to [74], initializing network with transferred features layers, followed by fine-tuning, brings improvement in deep neural network performance. Following this idea, in [92], child networks are forced to share weights so that the training costs can be significantly reduced.
- *Function-preserving transformation*: another line of research focus on the function-preserving transformation, first proposed in Net2Net [132] where new networks are initialized to represent the same functionality of a given trained model. This approach has been proven capable to significantly accelerate the training of the new network. Additionally, function-preserving transformation also inspires new strategies to explore the network architecture space in recent approaches [62], [72].

Summary. As we can observe, the applications of transfer learning in AutoML is relatively limited. Most approaches focused on neural network search problem, and the transferability of knowledge is not well addressed in an automatic manner, which motivates the discussion in Section 8.2.3.

7 Applications

In this section, we give details of the tree examples mentioned in Section 1. Besides, we will see how existing methods follow the basic framework of AutoML proposed in Section 2.2.2.

7.1 Model Selection using Auto-sklearn

As each learning problem has its own preference over learning tools [1], when we are dealing with a new classification problem, it is naturally to try a collection of classifiers and then get final predictions from an ensemble of them. This is a very typical application scenario of AutoML on model selection (Section 3.2), and the key issue here is how to automatically select best classifiers and setup their hyper-parameters.

Using Scikit-Learn [11] as an example, some popularly used classifiers and their hyper-parameters are listed in Table 4. In [5], [34], the above issue is considered as a CASH problem (Example 1). The ensemble construction is transferred into (2), which is an optimization problem minimizing the loss on validation set and involving with both parameters and hyper-parameters.

Example 1 (CASH Problem [5], [34]). *Let $\mathcal{F} = \{F_1, \cdots, F_R\}$ be a set of learning models, and each model has hyper-parameter θ_j with domain Λ_j, $\mathcal{D}_{train} = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)\}$ be a training set which is split into K cross-validation folds $\{\mathcal{D}^1_{train}, \cdots, \mathcal{D}^K_{train}\}$ and $\{\mathcal{D}^1_{valid}, \cdots, \mathcal{D}^K_{valid}\}$ with $\mathcal{D}^i_{train} \cup \mathcal{D}^i_{valid} = \mathcal{D}_{train}$ for $i = 1, \ldots, K$. Then, the Combined Algorithm Selection and Hyper-parameter (CASH) optimization problem is defined as*

$$F^*, \theta^* = \operatorname*{argmin}_{\substack{\theta \in \Lambda_j \\ F_j \in \mathcal{F}}} \frac{1}{K} \sum_{i=1}^{K} \min_{\mathbf{w}} \mathcal{L}\left(F_j(\mathbf{w}; \theta), \mathcal{D}^i_{train}, \mathcal{D}^i_{valid}\right), \quad (2)$$

where $\mathcal{L}\left(F_j(\mathbf{w}_j; \theta_j), \mathcal{D}^i_{train}, \mathcal{D}^i_{valid}\right)$ denotes the loss that F_j achieves on $\mathcal{D}^i_{valid}$ with parameter $\mathbf{w}_j$, hyper-parameter θ_j and training data $\mathcal{D}^i_{train}$.

Thus, the configuration here is made up by choices of classifiers in Scikit-Learn and their hyper-parameters, and the search space is spanned by such configuration. However, (2) is very hard to optimize. First, the objective function is defined on the validation set and we have little information

TABLE 10
Illustration of how examples in Table 1 fall into the proposed framework in Figure 6. The "naive" means there is no special design in the evaluator, the evaluation is directly done by optimizing parameters of learning tools on the training data.

example	controller		learning tools
	optimizer	evaluator	
Auto-sklearn [5]	SMAC [88] algorithm (warm-start by meta-learning)	direct evaluation (train model parameter with optimization algorithms)	out-of-box classifiers
NASNet [20]	recurrent neural networks (trained with REINFORCE algorithm [133])	direct evaluation (train child network with stochastic gradient descent)	convolutional neural networks
ExploreKit [9]	greedy search algorithm	classifiers trained with meta-features	subsequent learning models

about such objective. Then, the variables need to be optimized, i.e., θ and F_j, may not even by continuous, e.g., θ for kNN includes the number of nearest neighbors. Finally, in order to get information on validation set, we need to train the model F_j and update its parameter $\mathbf{w}_j$, which is usually very expensive.

In [34], sequential model-based algorithm configuration (SMAC) [88], a method of Bayesian optimization, was used as the optimizer to solve (2). Then, the basic method, i.e., direct evaluation was used as the evaluator. Finally, meta-learning was utilized as the advanced technique, which helped to get better initialization (i.e., warm-start in Section 6.1.3).

7.2 Reinforcement Learning for NAS

As mentioned in Section 1, since the success of AlexNet on image classification of ImageNet dataset [14], the change on the design of neural architectures has been the main point for getting better predicting performance. Thus, a very natural question is that: can we automatically design neural architectures so that the generated architecture can have good learning performance across various tasks and datasets [7], [12], [13].

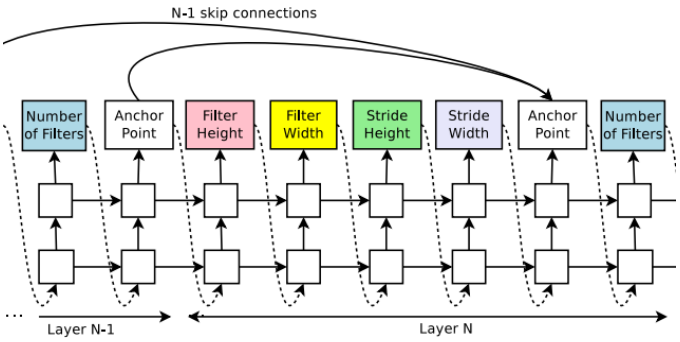


Fig. 18. Generating hyper-parameters for one convolutional layer using RNN (the image is from [20])

Taking generation of the convolutional layer in convolutional neural networks (CNN) as an example. For each convolution layer, basically, we need to determine (a). the number of filters, (b). filter height, (c). filter width, (d). stride height, (e). stride width and (f). skip connections (i.e., which output of previous layers should be taken as the input to current layer). An illustration of these designs is in Figure 10. Once these configurations are determined, one needs to train the network on the training data, and then adjust hyper-parameters based on the performance obtained from the validation set. The design choices and hyper-parameters are configurations.

However, CNN is naturally built in an incremental manner, i.e., data are processed by low-level first and then come up with higher levels. Thus, unlike the previous example, the number of parameters and hyper-parameters in CNN are unknown as the number of layers is not fixed. As shown in Figure 18, these motivate representing architecture of CNN as the sequential data, and learn with the recurrent neural network (RNN). Besides, as the search space can be incrementally built, reinforcement learning, which can capture such dynamic property, is used to train the RNN [7], [13]. The RNN acts as the optimizer, and the direct evaluation is used as the evaluator. Finally, as RL is slow to converge, to make the search faster, transfer learning, which is used to cut the search space into several blocks, was later developed in [21], [59].

7.3 Feature Construction using ExploreKit

One of the most representative works in the realm of automatic feature construction is ExploreKit [9]. It aims to generate new features such that the performance of subsequent learning tools can be improved. Figure 19 shows the system architecture of ExploreKit. The configuration here is a set of generated features. The main body is an iterative process, where each iteration comprises three steps, to search such configuration's space: candidate feature generation, candidate feature ranking, candidate feature evaluation and selection. Additionally, meta-learning techniques are employed in the ranking step to fast estimate the usefulness of candidate features and accelerate the subsequent evaluation step.

At the candidate feature generation step, new features are constructed by applying operators on features that are already selected. Employed operators include: 1) unary ones (e.g., discretization, normalization), 2) binary ones (e.g., $+$, $-$, $\times$, $\div$), and 3) higher-order ones (e.g., *GroupByThenMax*, *GroupByThenAvg*). A predetermined, enumerating procedure is invoked to apply these operators on all selected features that are applicable to generate candidates. In order to limit the size of candidate feature set, generated features will not be reused to further generate new candidates.

Since ExploreKit generates candidates exhaustively, evaluating all these features may be computational intractable. To address this issue, ExploreKit puts a ranking step before the evaluation and selection step. At this step, a ranking classifier, trained with historical knowledge on feature engineering, is used to fast rank the candidate features. In the evaluation step that follows, features predicted more useful will be considered first.

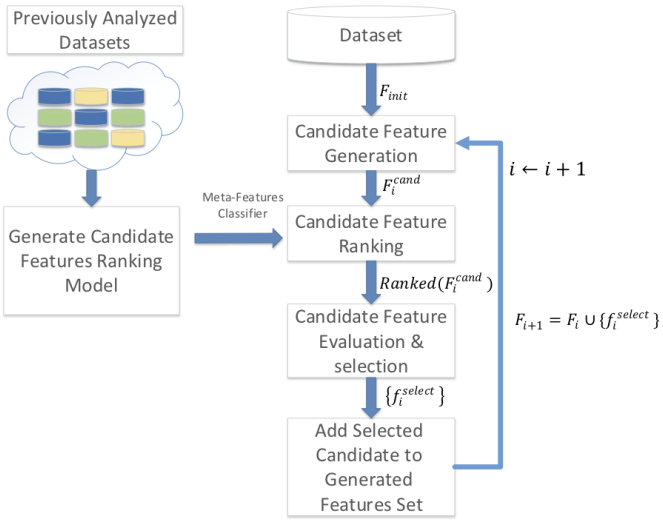


Fig. 19. The system architecture of ExploreKit (the image is from [9]).

Finally, ExploreKit conducts more expensive and accurate evaluations on the candidate features. Error reduction on the validation set is used as the metric for feature importance. Candidate features are evaluated according to their ranking, and selected if their usefulness surpass a threshold. This procedure terminates if enough improvement is achieved. The selected features will be used to generate new candidates in the following iterations.

8 Summary

In this section, we first review the history of AutoML (Section 8.1), then discuss its future works (Section 8.2). Finally, the survey is concluded in Section 8.3.

8.1 A Brief History of AutoML

While AutoML was recently mentioned in [34] and introduced in ICML-2014's AutoML workshop [134], as mentioned in the introduction (Section 1), such idea appears long time ago. However, the automation only recently becomes possible and a big focus, due to the big data, increasing computation of modern computers, and of course, great demand of machine learning application. One big picture of AutoML is shown in Figure 1, all of its components, i.e., feature engineering, model selection and algorithm selection, has been individually touched by many researchers in fields of machine learning, data mining and artificial intelligence.

On feature level, the feature selection is a traditional topic in machine learning, it ties to automatically remove unnecessary features making learning models simpler and more interpretable [135]. Many methods have been proposed and Lasso [136] is a landmark of them. Dimension reduction methods, such as Principal component analysis (PCA) [64] and Linear discriminant analysis (LDA) [65], have also been popularly used to deal with high-dimensional features. These methods tries to find a better representation for input features. However, the size of the reduced dimension needs human specifications. During 1990s, many attempts to automatic construction of better features were done based on genetic algorithms [67], [137], [138]. But, it is more recently that automatic learning feature representations became possible for some structured data, such as convolutional neural networks (CNN) for images and recurrent neural networks for (RNN) sequential data [63].

Model selection is the task of selecting a proper model from a set of candidate models, given data. Theoretical foundation, i.e., statistical learning theory [139], [140], is firstly paved for model selection, which shows how a model can generalize from the given to unseen data. For a specific model, its performance is most influenced by its hyper-parameters. The grid search is the most commonly used method to determine a proper value for hyper-parameters. However, it can only deal with a very small number of hyper-parameters. Later, optimization based methods, such as derivative-free optimization [141], [142] and gradient based optimization [93], have been considered for finding hyper-parameters. These methods have strong assumptions on the optimization model, and do not consider randomness in the learning problems. These motivates many tailor-made methods for the selection of various machine learning models, such as the kernel selection for SVM [143], learning K for K-means [144] and genetic programming for neural networks architectures [80], [82]. Finally, as a single model may not be strong enough, ensemble of models is considered in [145].

Finally, once the model is fixed, optimization algorithms are required to find good parameters. Algorithm selection originally dates back to 1970's, where many researcher tried to design better algorithms for hard combinatorial problems [51], [146]. As these problems are mostly NP-hard, it is not possible to find their optimal solutions. Each algorithm has its own heuristics and strength for solve a certain type of problems. The algorithm selection attempts to identify the best algorithm for the given combinatorial problem. Traditionally, convex and simple models, such as logistic regression and SVM, are used in machine learning. Besides, the data sets are also not large. Under such environment, algorithm selection is not an important problem, and it is easy to find good parameters for a given model [147]. Recently, as the data gets larger and complex models become more popular, it is usually not possible to find good parameters, the optimization algorithms can have important impact on the generalization performance [71], [148], [149].

Nowadays, AutoML is a very complex and active research area, there are many new opportunities and problems in AutoML that are not been toughed in the above history, e.g., usage with advanced techniques (Section 6). Lots of papers focusing on AutoML appear on various conferences and journals, such as ICML, NIPS, KDD, AAAI, IJCAI and JMLR; many workshops are organized, such as AutoML workshop at ICML from 2014 to 2018 [134], [150], [151], [152], [153]; and some competitions such as, AutoML Challenge at PAKDD [154] and NIPS [155], are hold as well.

8.2 Future Works

First of all, as AutoML focuses on how to do machine learning in a special way (Definition 2), the current trends in machine learning can also be seen as future works of

AutoML. Examples are human interpretability [156] and privacy [157] in machine learning. In the sequel, we focus more on future works which links closer to the framework in Section 2.2.

8.2.1 Efficiency

As the search space is very complex and the evaluation is very expensive, AutoML is an extremely resources-consuming problem. For example, 800 GPUs are used in [13] for NAS with reinforcement learning. Thus, developing more efficient AutoML approaches is always desired. These can either done by proposing algorithms for the optimizer, which needs consuming less configurations before reaching a good performance, or by designing better methods for the evaluator, which can offer more accurate evaluations but in less time. Currently, as mentioned in Section 6, some promising directions are using transfer learning or meta-learning, which can bring the benefits of the external knowledge, to enhance over basic techniques.

8.2.2 Problem setup

How to create features from the data is a fundamental problem not only in machine learning, but also appears in many related areas. For example, in computer vision, an early concept was Scale-Invariant Feature Transform (SIFT) [158] and histograms of oriented gradients (HoG) [159]. These types of features can be successfully generalized for many problems and applications in vision. Similarly, in natural language processing, there are generalized features generation techniques such as "term frequency-inverse document frequency (TF-IDF)" [160], which is the ratio of how frequently a word shows up in a document to how often it shows up in the whole corpus of documents. This feature type is easy to calculate and performs well.

As what data should be used for the subsequent learning tools heavily depends on application scenarios (Figure 1), there are no general rules or unified models for creating features from the data. Specifically, interactions underneath the given data need to be understood by humans, then features are usually designed based on humans' expertise, e.g., SIFT and TF-IDF. Due to such difficulties, automatically creating features from the data have only became possible for some specific data types. For example, SIFT has been replace by CNN and TF-IDF have been taken over by RNN. More recently, some endeavors have been made for relational dataset, i.e., DSM [10] (Table 2). Their success lies on utilizing the common relationships inside the data. Besides, with those automatic generated features, the performance of subsequent learning tools are significantly improved. Thus, one important future work is to automatic creating features from the data.

8.2.3 Techniques

Basic techniques: In Section 2.2, we have proposed a framework, where configurations are updated based on alternating iterations between the optimizer and evaluator. However, when data set is larger, it may not be efficient enough to update configurations only after parameters are identified by the evaluator. Such idea has been recently explored in [161] for finding continuous hyper-parameters in deep networks. One future direction is to simultaneously update both parameters (controlled by the evaluator) and configurations (controlled by the optimizer).

Advanced techniques: Meta-learning has been widely used to facilitate AutoML. However, there are some considerations for using meta-learning, which also indicate the orientation for future study, for example: how to better characterize learning problems, tools, and any other experience of interest; how to effectively and efficiently collect meta-knowledge; and how to study the reasons underneath the success or failure of a learning tool. Furthermore, we would like to point out that though meta-learning can help AutoML, how to automate meta-learning is also an interesting and meaningful research topic.

Transfer learning has found its successful application mainly in the realm of network architecture search. We are looking forward to more transfer learning techniques employed in a wider scope of AutoML, especially those to address the small-data problem, which has so far hindered the application of machine learning in many important and meaningful businesses, e.g., medical treatment. Also, it has been well realized that knowledge transfer does not always offer improvement. One topic of transfer learning is 'negative transfer' [26], where the phenomena that knowledge transfer causes performance degradation is studied. An appealing solution to address this issue is to automatically determine when and how to transfer what knowledge.

8.2.4 Applications

In this survey, we have focus on supervised-learning problems. They are also most considered ones in AutoML, due to the learning performance can be clearly evaluated. However, AutoML can be applied in many other problems in machine learning. For example, graph embedding [162], active learning [163], and semi-supervised learning [164]. Recently, there are some work in these directions [165]. While the framework in Section 2.2 can still be applied in these problems, as different learning tools have to be used, the search space is different. Besides, properties of these learning tools should be further explored in order to achieve good performance.

8.3 Conclusion

Motivated by the academic dream and industrial needs, the automatic machine learning (AutoML) has recently became a hot topic. In this survey, we give a systematical review of existing AutoML approaches. We first define what is the AutoML problem and then introduce a basic framework of how these approaches are realized. We also provide taxonomies of existing works based on "what" and "how", which can act as a guidance to design new and use old AutoML approaches. We further provide and discuss how existing works can be organized according to our taxonomies in details. Finally, we briefly review history of AutoML and show promising future directions. We hope this survey can act as a good guideline for beginners and show light upon future researches.

REFERENCES

[1] T. Mitchell, *Machine Learning*. Springer, 1997.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[4] W. Xiong, J. Droppo, X. Huang, F. Seide, M. L. Seltzer, A. Stolcke, D. Yu, and G. Zweig, "Toward human parity in conversational speech recognition," *IEEE/ACM Transactions on Audio, Speech and Language Processing*, vol. 25, no. 12, pp. 2410–2423, 2017.

[5] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems*, 2015, pp. 2962–2970.

[6] L. Kotthoff, C. Thornton, H. Hoos, F. Hutter, and K. Leyton-Brown, "Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA," *Journal of Machine Learning Research*, vol. 18, no. 1, pp. 826–830, 2017.

[7] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *International Conference on Learning Representations*, 2017.

[8] C. Liu, B. Zoph, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *European Conference on Computer Vision*, 2018.

[9] G. Katz, E. C. R. Shin, and D. Song, "Explorekit: Automatic feature generation and selection," in *International Conference on Data Mining*, 2016, pp. 979–984.

[10] J. M. Kanter and K. Veeramachaneni, "Deep feature synthesis: Towards automating data science endeavors," in *IEEE International Conference on Data Science and Advanced Analytics*, 2015, pp. 1–10.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[12] L. Xie and A. Yuille, "Genetic CNN," in *International Conference on Computer Vision*. IEEE, 2017, pp. 1388–1397.

[13] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *International Conference on Learning Representations*, 2017.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet: classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[15] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[18] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2261–2269.

[19] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 1998.

[20] I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le, "Neural optimizer search with reinforcement learning," 2017.

[21] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Conference on Computer Vision and Pattern Recognition*, 2017.

[22] M. J. Smith, R. Wedge, and K. Veeramachaneni, "FeatureHub: Towards collaborative data science," in *IEEE International Conference on Data Science and Advanced Analytics*, 2017, pp. 590–600.

[23] J. Vanschoren, "Understanding machine learning performance with experiment databases," *lirias. kuleuven. be, no. May*, 2010.

[24] C. Lemke, M. Budka, and B. Gabrys, "Metalearning: a survey of trends and technologies," *Artificial Intelligence Review*, vol. 44, no. 1, pp. 117–130, 2015.

[25] J. Vanschoren, "Meta learning," 2018. [Online]. Available: https://www.ml4aad.org/wp-content/uploads/2018/09/chapter2-metalearning.pdf

[26] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, no. 10, pp. 1345–1359, 2009.

[27] M. Feurer and F. Hutter, "Hyperparameter optimization," 2018. [Online]. Available: https://www.ml4aad.org/wp-content/uploads/2018/09/chapter1-hpo.pdf

[28] J.-H. M. Thomas Elsken and F. Hutter, "Neural architecture search," 2018. [Online]. Available: https://www.ml4aad.org/wp-content/uploads/2018/09/chapter3-nas.pdf

[29] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *arXiv preprint arXiv:1808.05377*, 2018.

[30] M. Guarnieri, "The roots of automation before mechatronics," *IEEE Industrial Electronics Magazine*, vol. 4, no. 2, pp. 42–43, 2010.

[31] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT Press, 2016.

[32] J. Rifkin, *The end of work: The decline of the global labor force and the dawn of the post-market era*. ERIC, 1995.

[33] C. L. Phillips and R. D. Habor, *Feedback control systems*. Simon & Schuster, 1995.

[34] C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown, "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2013, pp. 847–855.

[35] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures," in *International Conference Machine Learning*, 2013, pp. I–115.

[36] D. Maclaurin, D. Duvenaud, and R. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *International Conference on Machine Learning*, 2015, pp. 2113–2122.

[37] E. R. Sparks, A. Talwalkar, D. Haas, M. J. Franklin, M. I. Jordan, and T. Kraska, "Automating model search for large scale machine learning," in *ACM Symposium on Cloud Computing*, 2015, pp. 368–380.

[38] J. N. van Rijn, S. M. Abdulrahman, P. Brazdil, and J. Vanschoren, "Fast algorithm selection using learning curves," in *International symposium on intelligent data analysis*. Springer, 2015, pp. 298–309.

[39] Y. Yu, H. Qian, and Y.-Q. Hu, "Derivative-free optimization via classification." in *AAAI Conference on Artificial Intelligence*, vol. 16, 2016, pp. 2286–2292.

[40] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "SMASH: one-shot model architecture search through hypernetworks," in *International Conference on Learning Representations*, 2018.

[41] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[42] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[43] A. G. Baydin, B. A. Pearlmutter, A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," *Journal of Machine Learning Research*, vol. 18, pp. 1–43, 2017.

[44] R. Vilalta and Y. Drissi, "A perspective view and survey of meta-learning," *Artificial Intelligence Review*, vol. 18, no. 2, pp. 77–95, 2002.

[45] M. G. Smith and L. Bull, "Genetic programming with a genetic algorithm for feature construction and selection," *Genetic Programming and Evolvable Machines*, vol. 6, no. 3, pp. 265–281, 2005.

[46] B. Tran, B. Xue, and M. Zhang, "Genetic programming for feature construction and selection in classification on high-dimensional data," *Memetic Computing*, vol. 8, no. 1, pp. 3–15, 2016.

[47] F. Nargesian, H. Samulowitz, U. Khurana, E. B. Khalil, and D. Turaga, "Learning feature engineering for classification," in *International Joint Conference on Artificial Intelligence*, vol. 17, 2017, pp. 2529–2535.

[48] H. J. Escalante, M. Montes, and L. E. Sucar, "Particle swarm model selection," *Journal of Machine Learning Research*, vol. 10, no. Feb, pp. 405–440, 2009.

[49] V. Calcagno and C. de Mazancourt, "glmulti: An R package for easy automated model selection with (generalized) linear models," *Journal of Statistical Software*, vol. 34, no. i12, 2010.

[50] C. J. Merz, "Dynamical selection of learning algorithms," in *Learning from Data*. Springer, 1996, pp. 281–290.

[51] S. Kadioglu, Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann, "Algorithm selection and scheduling," in *International Conference on Principles and Practice of Constraint Programming*, 2011, pp. 454–469.

[52] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, "Algorithm runtime prediction: Methods & evaluation," *Artificial Intelligence*, vol. 206, pp. 79–111, 2014.

[53] B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Fréchette, H. Hoos, F. Hutter, K. Leyton-Brown, and K. Tierney, "ASlib: A benchmark library for algorithm selection," *Artificial Intelligence*, vol. 237, pp. 41–58, 2016.

[54] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves." 2015.

[55] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," in *International Conference on Learning Representations*, 2016.

[56] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter, "Towards automatically-tuned neural networks," in *Workshop on Automatic Machine Learning*, 2016, pp. 58–65.

[57] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra, "Pathnet: Evolution channels gradient descent in super neural networks," *arXiv preprint arXiv:1701.08734*, 2017.

[58] T. Elsken, J.-H. Metzen, and F. Hutter, "Simple and efficient architecture search for convolutional neural networks," *arXiv preprint arXiv:1711.04528*, 2017.

[59] Z. Zhong, J. Yan, and C.-L. Liu, "Practical network blocks design with Q-learning," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[60] B. Deng, J. Yan, and D. Lin, "Peephole: Predicting network performance before training," *arXiv preprint arXiv:1712.03351*, 2017.

[61] H. Jin, Q. Song, and X. Hu, "Efficient neural architecture search with network morphism," in *International Joint Conference on Artificial Intelligence*, 2018.

[62] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *AAAI Conference on Artificial Intelligence*, 2018.

[63] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[64] K. Pearson, "On lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[65] R. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[66] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *International Conference on Machine learning*, 2008, pp. 1096–1103.

[67] H. Vafaie and K. De Jong, "Genetic algorithms as a tool for feature selection in machine learning," in *International Conference on Tools with Artificial Intelligence*, 1992, pp. 200–203.

[68] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *IEEE Transactions on Image processing*, vol. 15, no. 12, pp. 3736–3745, 2006.

[69] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2010, pp. 2528–2535.

[70] K. Yu, T. Zhang, and Y. Gong, "Nonlinear learning using local coordinate coding," in *Advances in Neural Information Processing Systems*, 2009, pp. 2223–2231.

[71] L. Bottou and O. Bousquet, "The tradeoffs of large scale learning," in *Advances in Neural Information Processing Systems*, 2008, pp. 161–168.

[72] H. Cai, J. Yang, W. Zhang, S. Han, and Y. Yu, "Path-level network transformation for efficient architecture search," 2018.

[73] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[74] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in Neural Information Processing Systems*, 2014, pp. 3320–3328.

[75] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[76] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[77] A. R. Conn, K. Scheinberg, and L. N. Vicente, *Introduction to derivative-free optimization*. SIAM, 2009, vol. 8.

[78] T. Bck, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford Univ. Pr, 1998.

[79] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)," *Evolutionary Computation*, vol. 11, no. 1, pp. 1–18, 2003.

[80] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[81] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[82] C. Zhang, H. Shao, and Y. Li, "Particle swarm optimisation for evolving artificial neural network," in *IEEE International Conference on Systems, Man, and Cybernetics*, vol. 4, 2000, pp. 2487–2490.

[83] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *International Conference on Machine Learning*, 2017.

[84] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," *arXiv preprint arXiv:1802.01548*, 2018.

[85] T. Nickson, M. A. Osborne, S. Reece, and S. J. Roberts, "Automated machine learning on big data using stochastic algorithm tuning," *arXiv preprint arXiv:1407.7969*, 2014.

[86] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast bayesian optimization of machine learning hyperparameters on large datasets," in *International Conference on Artificial Intelligence and Statistics*, 2016.

[87] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, 2011, pp. 2546–2554.

[88] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International Conference on Learning and Intelligent Optimization*, 2011, pp. 507–523.

[89] Y.-Q. Hu, H. Qian, and Y. Yu, "Sequential classification-based optimization for direct policy search," in *AAAI Conference on Artificial Intelligence*, 2017, pp. 2029–2035.

[90] R. Munos, "Optimistic optimization of a deterministic function without the knowledge of its smoothness," in *Advances in Neural Information Processing Systems*, 2011, pp. 783–791.

[91] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6765–6816, 2017.

[92] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Faster discovery of neural architectures by searching for paths in a large model," in *International Conference on Learning Representations*, 2018.

[93] Y. Bengio, "Gradient-based optimization of hyperparameters," *Neural Computation*, vol. 12, no. 8, pp. 1889–1900, 2000.

[94] L. Franceschi, M. Donini, P. Frasconi, and M. Pontil, "Forward and reverse gradient-based hyperparameter optimization," in *International Conference on Machine Learning*, 2017, pp. 1165–1173.

[95] F. Pedregosa, "Hyperparameter optimization with approximate gradient," in *International Conference on Machine Learning*, 2016, pp. 737–746.

[96] A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter, "Learning curve prediction with bayesian neural networks," in *International Conference on Learning Representations*, 2016.

[97] K. Eggensperger, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Efficient benchmarking of hyperparameter optimizers via surrogates," 2015.

[98] K. A. Smith-Miles, "Cross-disciplinary perspectives on meta-learning for algorithm selection," *ACM Computing Surveys*, vol. 41, no. 1, p. 6, 2009.

[99] J. Gama and P. Brazdil, "Characterization of classification algorithms," in *Portuguese Conference on Artificial Intelligence*. Springer, 1995, pp. 189–200.

[100] S. Y. Sohn, "Meta analysis of classification algorithms for pattern recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 11, pp. 1137–1144, 1999.

[101] C. Taylor, D. Michie, and D. Spiegalhalter, "Machine learning, neural and statistical classifiers," 1994.

[102] P. Brazdil, J. Gama, and B. Henery, "Characterizing the applicability of classification algorithms using meta-level learning," in

European Conference on Machine Learning. Springer, 1994, pp. 83–102.

[103] C. Soares and P. B. Brazdil, "Zoomed ranking: Selection of classification algorithms based on relevant performance information," in *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 2000, pp. 126–135.

[104] H. Berrer, I. Paterson, and J. Keller, "Evaluation of machine-learning algorithm ranking advisors," in *PKDD-2000 Workshop on DataMining, Decision Support, Meta-Learning and ILP: Forum for Practical Problem Presentation and Prospective Solutions*. Citeseer, 2000.

[105] K. Alexandros and H. Melanie, "Model selection via meta-learning: a comparative study," *International Journal on Artificial Intelligence Tools*, vol. 10, no. 04, pp. 525–554, 2001.

[106] P. B. Brazdil, C. Soares, and J. P. Da Costa, "Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results," *Machine Learning*, vol. 50, no. 3, pp. 251–277, 2003.

[107] R. Leite and P. Brazdil, "Active testing strategy to predict the best classification algorithm via sampling and metalearning," in *European Conference on Artificial Intelligence*, 2010, pp. 309–314.

[108] M. Reif, F. Shafait, and A. Dengel, "Prediction of classifier training time including parameter optimization," in *Annual Conference on Artificial Intelligence*. Springer, 2011, pp. 260–271.

[109] N. Jankowski and K. Grabczewski, "Universal meta-learning architecture and algorithms," in *Meta-Learning in Computational Intelligence*. Springer, 2011, pp. 1–76.

[110] N. Jankowski, "Complexity measures for meta-learning and their optimality," in *Algorithmic Probability and Friends. Bayesian Prediction and Artificial Intelligence*. Springer, 2013, pp. 198–210.

[111] C. Soares, P. B. Brazdil, and P. Kuba, "A meta-learning method to select the kernel width in support vector regression," *Machine Learning*, vol. 54, no. 3, pp. 195–209, 2004.

[112] S. Ali and K. A. Smith-Miles, "A meta-learning approach to automatic kernel selection for support vector machines," *Neurocomputing*, vol. 70, no. 1-3, pp. 173–186, 2006.

[113] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning*, 2017, pp. 1126–1135.

[114] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *International Conference on Machine Learning*, 2018, pp. 549–558.

[115] T. A. Gomes, R. B. Prudêncio, C. Soares, A. L. Rossi, and A. Carvalho, "Combining meta-learning and search techniques to select parameters for support vector machines," *Neurocomputing*, vol. 75, no. 1, pp. 3–13, 2012.

[116] P. B. de Miranda, R. B. Prudêncio, A. C. P. de Carvalho, and C. Soares, "An experimental study of the combination of meta-learning with particle swarm algorithms for svm parameter selection," in *International Conference on Computational Science and Its Applications*. Springer, 2012, pp. 562–575.

[117] M. Reif, F. Shafait, and A. Dengel, "Meta-learning for evolutionary parameter optimization of classifiers," *Machine Learning*, vol. 87, no. 3, pp. 357–380, 2012.

[118] M. Feurer, J. T. Springenberg, and F. Hutter, "Initializing bayesian hyperparameter optimization via meta-learning." in *AAAI Conference on Artificial Intelligence*, 2015, pp. 1128–1135.

[119] M. Lindauer and F. Hutter, "Warmstarting of model-based algorithm configuration," in *AAAI Conference on Artificial Intelligence*, 2018, pp. 1355–1362.

[120] H. Hoos and K. Leyton-Brown, "An efficient approach for assessing hyperparameter importance," in *International Conference on Machine Learning*, 2014, pp. 754–762.

[121] J. N. van Rijn and F. Hutter, "Hyperparameter importance across datasets," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2018, pp. 2367–2376.

[122] M. Wistuba, N. Schilling, and L. Schmidt-Thieme, "Hyperparameter search space pruning–a new component for sequential model-based hyperparameter optimization," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2015, pp. 104–119.

[123] G. Widmer, "Tracking context changes through meta-learning," *Machine Learning*, vol. 27, no. 3, pp. 259–286, 1997.

[124] D. Kifer, S. Ben-David, and J. Gehrke, "Detecting change in data streams," in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. VLDB Endowment, 2004, pp. 180–191.

[125] R. Klinkenberg, "Meta-learning, model selection, and example selection in machine learning domains with concept drift." in *LWA*, vol. 2005, 2005, pp. 164–171.

[126] A. L. D. Rossi, A. C. Carvalho, and C. Soares, "Meta-learning for periodic algorithm selection in time-changing data," in *Brazilian Symposium on Neural Networks*, 2012, pp. 7–12.

[127] A. L. D. Rossi, A. C. P. de Leon Ferreira, C. Soares, and B. F. de Souza, "MetaStream: A meta-learning based method for periodic algorithm selection in time-changing data," *Neurocomputing*, vol. 127, pp. 52–64, 2014.

[128] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google vizier: A service for black-box optimization," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1487–1495.

[129] D. Yogatama and G. Mann, "Efficient transfer learning method for automatic hyperparameter tuning," in *Artificial Intelligence and Statistics*, 2014, pp. 1077–1085.

[130] K. Swersky, J. Snoek, and R. P. Adams, "Multi-task bayesian optimization," in *Advances in Neural Information Processing Systems*, 2013, pp. 2004–2012.

[131] C. Wong, N. Houlsby, Y. Lu, and A. Gesmundo, "Transfer automatic machine learning," 2018.

[132] T. Chen, I. Goodfellow, and J. Shlens, "Net2Net: Accelerating learning via knowledge transfer," in *International Conference on Learning Representations*, 2015.

[133] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[134] "ICML-2014 AutoML workshop," 2014. [Online]. Available: https://sites.google.com/site/automlwsicml14/

[135] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[136] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least angle regression," *Annals of Statistics*, vol. 32, no. 2, pp. 407–499, 2004.

[137] W. Tackett, "Genetic programming for feature discovery and image discrimination," in *International Conference on Genetic Algorithms*, 1993.

[138] W. Siedlecki and J. Sklansky, "A note on genetic algorithms for large-scale feature selection," in *Handbook of Pattern Recognition and Computer Vision*. World Scientific, 1993, pp. 88–107.

[139] V. Vapnik, *The nature of statistical learning theory*. Springer, 1995.

[140] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer Series in Statistics, 2001.

[141] M. J. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *The Computer Journal*, vol. 7, no. 2, pp. 155–162, 1964.

[142] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.

[143] N.-E. Ayat, M. Cheriet, and C. Y. Suen, "Automatic model selection for the optimization of svm kernels," *Pattern Recognition*, vol. 38, no. 10, pp. 1733–1745, 2005.

[144] G. Hamerly and C. Elkan, "Learning the k in k-means," in *Advances in Neural Information Processing Systems*, 2004, pp. 281–288.

[145] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, "Ensemble selection from libraries of models," in *International Conference on Machine Learning*, 2004, p. 18.

[146] J. Rice, "The algorithm selection problem," *Advances in Computers*, vol. 15, p. 65, 1976.

[147] S. Sra, S. Nowozin, and S. Wright, *Optimization for machine learning*. MIT Press, 2012.

[148] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[149] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, "On optimization methods for deep learning," in *International Conference on International Conference on Machine Learning*, 2011, pp. 265–272.

[150] "ICML-2015 AutoML workshop," 2015. [Online]. Available: https://sites.google.com/site/automlwsicml15/

[151] "ICML-2016 AutoML workshop," 2016. [Online]. Available: https://sites.google.com/site/automl2016/

[152] "ICML-2017 AutoML workshop," 2017. [Online]. Available: https://sites.google.com/site/automl2017icml/

[153] "ICML-2018 AutoML workshop," 2018. [Online]. Available: https://www.4paradigm.com/competition/pakdd2018
[154] "PAKDD 2018 data competition: Automatic machine learning challenge 2018," 2018. [Online]. Available: https://sites.google.com/site/automl2018icml/
[155] "NIPS 2018 challenge: AutoML for lifelong machine learning," 2018. [Online]. Available: https://sites.google.com/site/automl2018icml/
[156] "ICML-2018 workshop on human interpretability in machine learning," 2018. [Online]. Available: https://sites.google.com/view/whi2018/home
[157] "ICML-2018 workshop on privacy in machine learning and artificial intelligence," 2018. [Online]. Available: https://pimlai.github.io/pimlai18/
[158] D. G. Lowe, "Object recognition from local scale-invariant features," in *IEEE International Conference on Computer Vision*, vol. 2, 1999, pp. 1150–1157.
[159] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, 2005, pp. 886–893.
[160] G. Salton, A. Wong, and C.-S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.
[161] A. G. Baydin, R. Cornish, D. M. Rubio, M. Schmidt, and F. Wood, "Online learning rate adaptation with hypergradient descent," *arXiv preprint arXiv:1703.04782*, 2017.
[162] H. Cai, V. W. Zheng, and K. Chang, "A comprehensive survey of graph embedding: problems, techniques and applications," *IEEE Transactions on Knowledge and Data Engineering*, 2018.
[163] B. Settles, "Active learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 6, no. 1, pp. 1–114, 2012.
[164] X. Zhu, "Semi-supervised learning literature survey," *Computer Science, University of Wisconsin-Madison*, vol. 2, no. 3, p. 4, 2006.
[165] M. Fang, Y. Li, and T. Cohn, "Learning how to active learn: A deep reinforcement learning approach," in *Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 595–605.