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

Quanming Yao, Mengshuo Wang,

Yuqiang Chen, Wenyuan Dai, 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 pursue good learning performance, humans are heavily involved in every aspect of machine learning. To make machine learning

techniques easier to apply and reduce the demand for experienced human experts, automated machine learning (AutoML) has

emerged as a hot topic with both industrial and academic interest. In this paper, we provide an up to date survey on AutoML. 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 most existing approaches to date, but also can guide the design for new methods.

arXiv: 1810. 13306v3 [cs.AI] 17 Jan 2019

Subsequently, we categorize and review the existing works from two aspects, i.e., the problem setup and the employed techniques.

The proposed framework and taxonomies 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 research.

**Index Terms**—automated machine learning,neural architecture search,hyper-parameter optimization,meta-learning,transfer-learning  F

**1 INTRODUCTION**

[Mitchell’s famous machine learning textbook [1] begins with](#bookmark1) the statement: “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 quest gave birth to a new research area, i.e., machine learning, for Computer Science decades ago. 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, machine learning has also gained signiﬁcant achievements. For ex- [ample, AlphaGO [2] defeated human champion in the game](#bookmark2) [of GO, ResNet [3] surpassed human performance in image](#bookmark3) [recognition, Microsoft’s speech system [4] approximated](#bookmark4) human level in speech transcription.

However, these successful applications of machine learn- ing are far from fully automated, i.e., “improving auto- matically with experience ” . Since there are no algorithms that can achieve good performance on all possible learning problems with equal importance (according to No Free [Lunch theorems [5] [6]), every aspect of machine learning](#bookmark6) applications, such as feature engineering, model selection, and algorithm selection (Figure [1), needs to be carefully](#bookmark7) conﬁgured. Human experts are hence heavily involved in machine learning applications. As these experts are rare

• Q. Yao, M. Wang, Y. Chen and W. Dai are with 4Paradigm Inc, Beijing, China; Y. Hu, Y. Li and Y. Yu are with Nanjing uiversity, Jiangsu, China; and Q. Yang is with Hong Kong University of Science and Technology, HongKong, China.

• All authors are in alphabetical order of last name (except the ﬁrst two). Correspondance to Q. Yao at yaoquanming@4paradigm.com

resources, the success of machine learning comes at a great price.

Thus, automated machine learning (AutoML) does not just remain an academic dream as described in Michell’s book, but also attracts more attention from practitioners. If we can take the human out of these machine learning applications, we can enable faster deployment of machine learning solutions across organizations, efﬁciently validate and benchmark the performance of deployed solutions, and make experts focus more on problems with more application and business values. These would make machine learning much more accessible 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, AutoML has emerged as a new sub-area in machine learning. It has got more attention not only in machine learning but also in computer vision, data mining and natural language processing. Up to now, AutoML has already been successfully applied in many important problems (Table [1)](#bookmark5).

TABLE 1

Examples of AutoML approaches in industry and academic.

|  |  |  |
| --- | --- | --- |
| application | industry | academic |
| automated modelselection | Auto-sklearn | [[7],](#bookmark8) [[8]](#bookmark9) |
| neural architecture search | Google’s Cloud | [[9],](#bookmark10) [[10]](#bookmark11) |
| automated feature engineering | Feature Labs | [[11],](#bookmark12) [[12]](#bookmark13) |

[The ﬁrst example is Auto-sklearn [7]](#bookmark8). As different classi- [ﬁers are applicable to different learning problems [1], [13], it](#bookmark1) is natural to try a collection of classiﬁers on a new problem, and then construct a ﬁnal prediction from them. However, setting up classiﬁers and their hyper-parameters is a tedious task, which usually requires human involvement. Based on [the popular scikit-learn machine learning library [14], Auto](#bookmark14)-

2

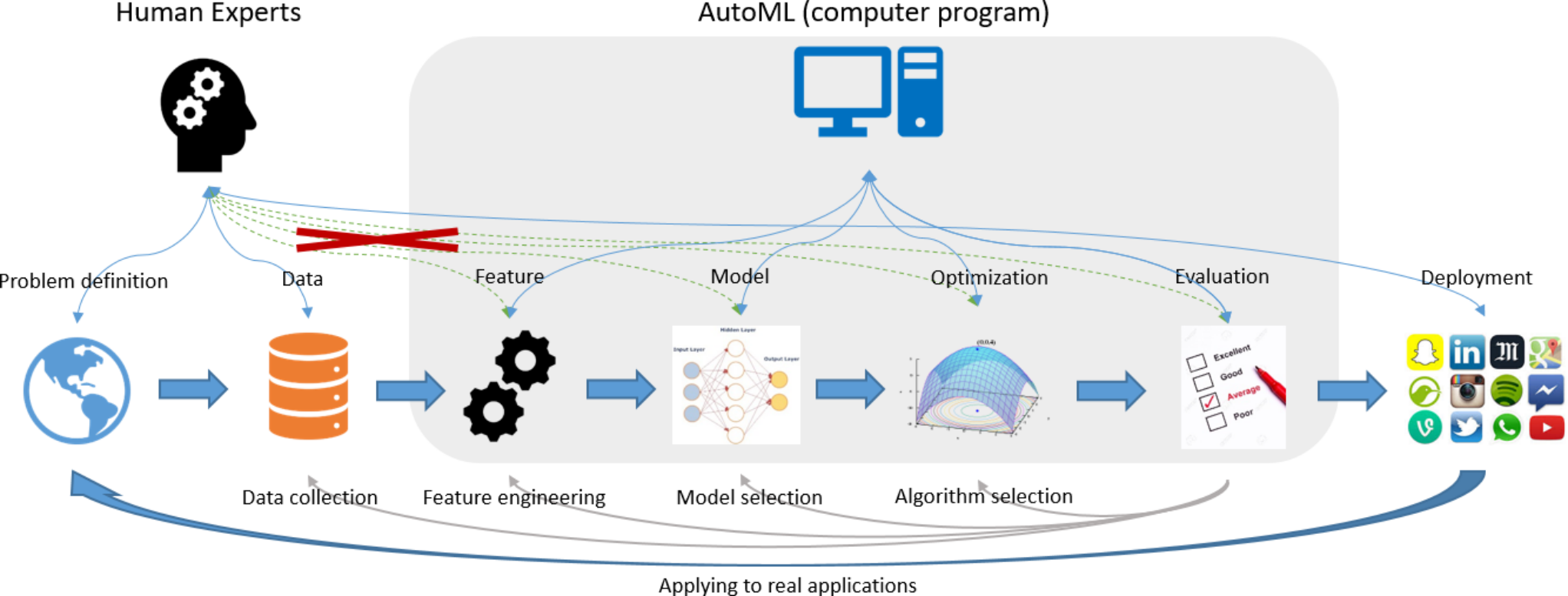


Fig. 1. To use machine learning techniques and obtain good performance, humans usually need to be involved in data collection, feature engineering, model and algorithmselection. This picture shows a typical pipeline of machine learning application, and how AutoML can get involved in the pipeline and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

sklearn can automatically ﬁnd good models from some out-of-the-box machine learning tools for classiﬁcation by searching for proper models and optimizing their corre- spondinghyper-parameters.

The second example is the neural architecture search [(NAS) [9], [15], [16]](#bookmark10). [Since the success of AlexNet [17] on](#bookmark16) [image classiﬁcation of ImageNet data set [18], architecture](#bookmark17) design has become the main source of performance im- provement in the realm of deep learning. Examples are [VGGNet [19], GoogleNet [20], ResNet [3] and DenseNet](#bookmark18) [[21]](#bookmark19). Hence, for the tasks in hand, automated design of neural architectures is of great importance to good learning performance. Many researchers have been working on NAS, [e.g., [9], [10], [16], [22], [23], [24], [25], [26]](#bookmark10). Besides, NAS has been used in Google’s Cloud AutoML, which frees cus- tomers from the difﬁcult and time-consuming architecture design process.

The last example is automated feature engineering. In traditional machine learning methods, the modeling perfor- [mance depends greatly on the quality of features [1]](#bookmark1). Hence, most machine learning applications take feature engineering as a vital preposition step, where useful features are gener- ated or selected. Such operations, in the past, are usually car- ried out manually by human experts within-depth domain knowledge in a trial-and-error manner. Automated feature [engineering [11], [12] aims to construct a new features](#bookmark12) 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) [12],](#bookmark13) [ExploreKit [11] and FeatureHub [27]](#bookmark12). Besides, we have also [seen commercial products such as FeatureLabs [12]](#bookmark13).

With such a rapid development of AutoML in both research and industry, we feel it necessary to summarize existing works and conduct a survey on this topic at this moment. First, we discuss what the AutoML problem is. Then, we propose a general framework that summarizes how existing approaches work towards AutoML. Such a framework further motivates us to give taxonomies of exist-

ing works based on what (by problem setup) and how (by techniques) to automate. Speciﬁcally, problem setup helps us to clarify what learning process we want to use, while techniques give us the technical methods and details toad- dress the AutoML problem under the corresponding setup. Based on these taxonomies, we further give a guidance on how AutoML approaches can be developed.

**1.1 Contributions**

Below, we summarize our contributions in this survey:

• We discuss the formal deﬁnition of AutoML. The deﬁni- tion is not only 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 set- ting up taxonomies of existing works, but also gives in- sights of the problems existing approaches want to solve. Such framework can act as a guidance for developing new approaches.

• We systematically categorize existing AutoML works based on “what to automate” and “how to automate” . Problem setups are from the “what” perspective, indicat- ing which learning process we want to make automated. Techniques are from the “how” perspective, introducing the methods proposed to solve AutoML problems. For each category, we present detailed application scenarios for reference.

• [Compared to existing AutoML related surveys1,](#bookmark20) we pro- vide a detailed analysis of existing techniques, which is based on the proposed framework. We not only in- vestigate a more comprehensive set of existing works, but also present a summary of the insights behind each

1. In this survey we focus on the usage of existing techniques in AutoML, for individual reviews on related topics please refer to [[28],](#bookmark21)

[[29],](#bookmark22) [[30] for meta-learning,](#bookmark23) [[31] for transfer learning,](#bookmark24) [[32] for hyper](#bookmark25)- parameter optimization and [[33] for neural architecture search](#bookmark26).

3

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 the problem setting, techniques, applications and theory. For each, we provide a thorough analysis of its disadvantages in the current work and propose future research directions.

**1.2 Organization**

The survey is organized as follows. The overview is in Sec- tion [2, which gives the deﬁnition of AutoML, the proposed](#bookmark28) framework of AutoML approaches, and taxonomies by problem setup and techniques of existing works. Section [3](#bookmark29) describes the taxonomy by problem setup, and techniques are detailed in Section [4-6.](#bookmark30) Three application examples listed in Table [1](#bookmark5) are detailed in Section [7.](#bookmark31) The survey is summa- rized in Section [8](#bookmark32)with a brief history, the current status, and discussion on future works. Finally, we conclude the survey in Section [9.](#bookmark33)

**1.3 Notation**

In the rest of this survey, we denote a machine learning tool as F (**x**; θ), where **x**is the model parameters learned by training and θ contains conﬁgurations of the learning tool. Besides, the most important concepts used in the survey are explained as follows.

• A **learning process** is a part or the whole of a machine learning pipeline. Examples of learning processes are feature engineering, model and/or algorithm selection, and neural architecture design.

• A **learning tool** is a method which can solve some prob- lems appear in machine learning. For example, a support vector machine (SVM) model is a learning tool, which can solve speciﬁc classiﬁcation problems; and sparse coding

[[34] is also a learning tool, which can address feature](#bookmark35) learning problem for certain types of data.

• We use the term **conﬁguration** to denote all factors but the model parameters **x** (which are usually obtained from model training) that inﬂuence the performance of a learn- ing tool. Examples of conﬁgurations are, the hypothesis class of a model, the features utilized by the model, hyper-parameters that control the training procedure, and the architecture of a neural network.

**2 OVERVIEW**

In Section [1, we have shown why we need to do AutoML](#bookmark15). In this section, we ﬁrst deﬁne what AutoML problem is in Section [2.1.](#bookmark36) Then, in Section [2.2](#bookmark37) 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.](#bookmark38)

**2.1 Problem Deﬁnition**

Inspired by automation and machine learning, here, we deﬁne what the AutoML problem is. 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 [35], machine learning was only invented decades ago](#bookmark41) [[1]](#bookmark1). 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](#bookmark27) 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])](#bookmark1)**.** 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.

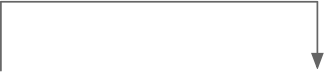
From this perspective, AutoML itself can also be seen as a learning tool that has good generalization performance (i.e., P) on the input data (i.e., E) and given tasks (i.e., T). However, traditional machine learning researches focus more on inventing and analyzing learning tools, it does not care much 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 [36]](#bookmark42). In the contrast, AutoML emphasizes on how easy learning tools can be used. This idea is illustrated in Figure [2.](#bookmark34)

**AutoML: computer program**



|  |  |
| --- | --- |
| |  | | --- | | **a strong and powerful learning tool** | |

**self-adapting**



**to problems**



|  |
| --- |
| **prediction** |



|  |
| --- |
| **data** |

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 [[37]](#bookmark43). In pursuit of better predicting performance, conﬁg- urations 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](#bookmark39) from this perspective is to construct high-level controlling approaches over underneath learning tools so that proper conﬁgurations can be found without human assistance.

**AutoML: computer program**



|  |
| --- |
| **data** |

|  |
| --- |
| **prediction** |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **controller** |   **conﬁgurations**   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  |  | | **feature** | | **model** | | **algorithm** | | |

Fig. 3. AutoML from automation’s perspectives.

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

4

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 and construct features from data | automated by the computer program |
| humans process features making them more informative |
| modelselection | 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 |
| algorithmselection | humans pick up some optimization algorithms to ﬁnd parameters |
| summary | human experts are involved in every aspect of machine learning applications | the program can be directly reused on other learning problems |

*2.1.2 The Deﬁnition of AutoML*

From Section [2.1.1, we can see that AutoML not only wants](#bookmark40) to have good learning performance (from machine learn- ing’s perspective) but also requires such performance be- ing achieved without human assistance (from automation’s perspective). Thus, an informal and intuitive description of AutoML can be expressed as

max performance of learning tools (1)

conﬁgrations ,

s.t. {al budget .

Put it more formally, we describe what AutoML is in Deﬁnition [2.](#bookmark45) Such deﬁnitionis inspired by Deﬁnition [1](#bookmark27) and the fact that AutoML itself can also be seen as another machine learning approach (Figure [2)](#bookmark34).

**Deﬁnition 2** (AutoML)**.** AutoML attempts to construct ma- chine learning programs (speciﬁed by E, T and P in Deﬁni- tion [1), without human assistance and within limited computa](#bookmark27)- tional budgets.

A comparison of classical machine learning and Au- toML is in Table [2.](#bookmark44) Basically, in classical machine learning, human are heavily involved in conﬁguring learning tools by operating feature engineering, model selection and al- gorithm selection. 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 Deﬁnition [2](#bookmark45)better, let us look back at those three examples in Table [1:](#bookmark5)

• Automated feature engineering: When original features are not 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 [12] and ExploreKit](#bookmark13)

[[11] remove human assistance by automatically construct](#bookmark12) new features based on interaction among input features.

• Automated modelselection: Here, E denotes input training data, T is a classiﬁcation task, and P is the performance 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 (NAS): 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 performance 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.

Finally, note that Deﬁnition [2](#bookmark45)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 will not be further pursuit in the sequel.

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 computa- tional budgets. These set up three main goals for AutoML (Remark [2.1)](#bookmark46).

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

(A). Good performance: good generalization performance across various input data and learning tasks can be achieved;

(B). No assistance from humans: conﬁgurations can beautomat- ically done for machine learning tools; and

(C). High computational efﬁciency: the program can return an reasonable output within a limited budget.

Since AutoML itself can be seen as a machine learning tool (Figure [2), here we remark that the goal (A) actually](#bookmark34) intends to escape the “curse” of the notorious No Free [Lunch theorems stated in [5] and [6]](#bookmark6). These theorems state that in a noise-free scenario of supervised learning, all learn- ing algorithms have the same generalization performance (error rate) when averaged over all possible learning tasks. Although these theorems are mathematically proven, it is hard (and even impossible) to apply them to the reality and make empirical test. This is because that the average on the performance over all possible learning tasks (with equal weights) is very brutal. It is highly possible that learning tasks in reality take up only a very narrow spectrum in all theoretically possible tasks. And the information on the distribution of reality tasks can be utilized in a context of meta-learning or transfer learning, for which we will give further investigation in Section [6.1](#bookmark47) and [6.2.](#bookmark48)

Once above three goals can be realized, we can fast deploy machine learning solutions across organizations, quickly validate and benchmark the performance of de- ployed solutions, and let human focus more on problems that really need humans’ engagements, i.e., problem deﬁni-

5

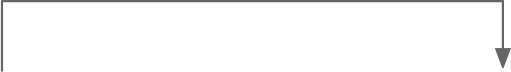
tion, data collection and deployment in Figure [1.](#bookmark7) All these make machine learning easier to apply and more accessible for everyone.

**2.2 Basic Framework**

In Section [2.1, we have deﬁned the AutoML problem (Def](#bookmark36)- inition [2) and introduced its core goals (Remark](#bookmark45) [2.1)](#bookmark46). In this section, we propose a basic framework for AutoML approaches.

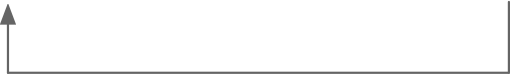
*2.2.1 Human Tuning Process*

However, before that, let us learn how conﬁgurations are tuned by human. Such process is shown in Figure [4.](#bookmark49) 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,](#bookmark49) i.e., feature, model or optimization in Figure [1.](#bookmark7) To obtain a good learning performance, we will try to set a conﬁgura- tion 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.



**try a conﬁguration**

|  |
| --- |
| **learning tools** |





|  |
| --- |
| **human expert** |

**feedback**

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

*2.2.2 Proposed AutoML Framework*

Motivated by the human-involved process above and con- [trolling with feedbacks in the automation [38], we summa](#bookmark51)- rize a framework for AutoML, as shown in Figure [6.](#bookmark52) Com- pared with Figure [4, in this ﬁgure, an AutoML controller](#bookmark49) 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](#bookmark52) 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 feed- backs to the optimizer. Usually, to measure the perfor- mance of learning tools with given conﬁguration, the evaluator needs to train a 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 maybe inaccurate. Thus, for the evaluator, it needs to beefﬁcient but also accurate in measuring the performance of conﬁgurations.

• Optimizer: Then, for the optimizer, its duty is to update or generate conﬁgurations for learning tools. The search space

of the optimizer is determined by the targeted learning process, and new conﬁgurations are expected to have bet- ter performance than previous ones. However, feedbacks offered by the evaluator are not necessarily required or exploited by the optimizer. This depends on which type of the optimizer we are utilizing. Finally, the optimizer should be chosen based on the learning process and corresponding search space, as the latter determines the applicability of different optimization methods. We also wish the structure of the search space can be simple and compact so that more generic and efﬁcient optimization methods can be employed.

As we will see, this framework is general enough to cover nearly all existing works (e.g. [[7], [9], [11], [12], [22],](#bookmark8) [[39], [40], [41], [42], [43], [44], [45], [46], just to name a few)](#bookmark53). In Section [7, we provide more detailed examples demon](#bookmark31)- strating how the it can cover existing works. Furthermore, this framework is also precise enough to help us setup taxonomies for AutoML approaches (Section [2.3), and it](#bookmark38) gives insight to the future direction of AutoML (Section [8.3)](#bookmark54).

**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](#bookmark55) make automated by AutoML. Basically, for general learning problems, we need to do feature engineering, model selec- tion and optimization algorithmselection. These three parts together make up the full scope of general machine learning applications (Figure [1)](#bookmark7). We also list 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.](#bookmark29)

*2.3.2 “How to automate”: by techniques*

Figure [5(b)](#bookmark56) presents the taxonomy by AutoML techniques. These are the techniques used for the controller, and cate- gorize “how” we solve an AutoML problem. In general, we divide existing techniques into basic and experienced ones: • Basic techniques: As there are two ingredients,i.e., the op- timizer and evaluator, in the controller, we categorize ba- sic 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 [[47] to more complex ones as reinforcement learning](#bookmark57)

[[9] and automatic differentiation [48]](#bookmark10). 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.

• Experienced techniques: Experienced techniques learn and accumulate knowledge from past searches or external data. They usually need to be combined with basic tech- niques and enhance the optimizer and/or evaluator in

6

**full scope: general case**

|  |
| --- |
|  |
|  |
| **full** |

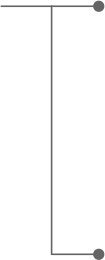
|  |
| --- |
| **problem setup** |

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | **feature engineering** |  |  | | --- | | **model selection** |  |  | | --- | | **algorithm selection** | |

**scope: deep learning**

|  |  |
| --- | --- |
| |  | | --- | | **neural architecture search** | |

(a) “What to automate”: by problem setup.

**optimizer: optimization & search**



|  |
| --- |
| **basic** |



|  |
| --- |
| **techiniques** |

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | **simple search approaches** |  |  | | --- | | **optimization from samples** |  |  | | --- | | **gradient descent** | |

**evaluator: performance evaluation**

|  |
| --- |
| **experienced** |

|  |  |
| --- | --- |
| |  | | --- | | **basic evaluation strategies** | |

|  |
| --- |
| **meta-learning** |

|  |
| --- |
| **transfer learning** |

(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.](#bookmark52) 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. Speciﬁcally, feature engineering, model selection and optimization algorithm selection together make up the full scope of general machine learning applications (Figure [1)](#bookmark7).

**AutoML controller**

|  |
| --- |
| **learning process**  feature, model, algorithm |

**determine**

**search space**

|  |
| --- |
| **learning tools** |

**measure**

**performance**

|  |  |  |
| --- | --- | --- |
| **feedback**  **conﬁguration**   |  | | --- | | **optimizer** |  |  | | --- | | **evaluator** | |

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

various manners. Generally, there are two main methods [popularly used in AutoML, i.e., meta-learning [29], [49]](#bookmark22) [and transfer learning [31]](#bookmark24).

Note that, as E, T and P are also involved in the AutoML’s deﬁnition (Deﬁnition [2), taxonomies of machine](#bookmark45) learning, e.g., supervised learning, semi-supervised learning and unsupervised learning, can also be applied for AutoML. However, they do not necessarily connect with removing human assistance in ﬁndingconﬁgurations (Figure [4)](#bookmark49). Thus, taxonomies here are done based on the proposed framework in Figure [6](#bookmark52)instead. Finally, we focus on supervised AutoML approaches in this survey as all existing works for AutoML are supervised ones.

*2.3.3 Workﬂow based on Taxonomies*

In the sequel, basic techniques and core issues they need to solve are introduced in Section [4](#bookmark30)and [5](#bookmark58)for the optimizer and evaluator respectively. After that experienced techniques are described in Section [6.](#bookmark59) The working ﬂow of designing an AutoML approach is summarized in Figure [7, which also](#bookmark60) acts a a guidance through this survey.

**problem setup techniques**

**AutoML controller**

**Section 3**

|  |
| --- |
| **setup**  ﬁgure out *conﬁgurations*, determine *search space* |

|  |  |  |
| --- | --- | --- |
| |  | | --- | | **optimizer**  ﬁnd basic methods for the *optimizer* |   **Section 4**   |  | | --- | | **evaluator**  ﬁnd basic methods for the *evaluator* |   **Section 5** |



|  |
| --- |
| **experienced techniques**  further improve  performance and eﬃciency |

**Section 6**

Fig. 7. Working ﬂow of designing AutoML approaches based on the proposed framework (Figure [6) and taxonomies](#bookmark52).

**3 PROBLEM SETTINGS**

In this section, we give details on categorization based on problem setup (Figure [5(a))](#bookmark55). Basically, it clariﬁes what to be automated. AutoML approaches do not necessarily cover the full machine learning pipeline in Figure [1, they can also](#bookmark7) focus on some parts of the learning process. In order to setup an AutoML problem, common questions that should be asked are:

**Remark 3.1.** Three important questions to setup an AutoML problem are

(A). What learning process we want to focus on?

(B). What learning tools can be designed and used?

(C). What are resultant corresponding conﬁgurations?

By answering these questions we can deﬁne the search space for an AutoML approach. Table [3](#bookmark62) gives an overview on how the focused learning process changes the search space. In the sequel, we brieﬂy summarize existing learning tools for each setup and what are the corresponding search space.

7

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)](#bookmark61).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| learning process | | learning tools | search space | examples |
| feature engineering | | (subsequent) classiﬁers | feature sets | [[11],](#bookmark12) [[12],](#bookmark13) [[50],](#bookmark63) [[51],](#bookmark64) [[52]](#bookmark65) |
| feature enhancing methods and their hyper-parameters | [[7],](#bookmark8) [[8],](#bookmark9) [[39]](#bookmark53) |
| modelselection | | classiﬁers | classiﬁers and their hyper-parameters | [[7],](#bookmark8) [[39],](#bookmark53) [[42],](#bookmark66) [[53],](#bookmark67) [[54]](#bookmark68) |
| optimization algorithmselection | | classiﬁers | algorithms and their hyper-parameters | [[43],](#bookmark69) [[55],](#bookmark70) [[56],](#bookmark71) [[57],](#bookmark72) [[58]](#bookmark73) |
| full scope | general | classiﬁers | an union of search space in feature, model, and/or algorithm | [[7],](#bookmark8) [[8],](#bookmark9) [[23],](#bookmark74) [[39]](#bookmark53) |
| neural architecture  search (NAS) | neural networks | network structures | [[9],](#bookmark10) [[22],](#bookmark75) [[25],](#bookmark76) [[45],](#bookmark77) [[59],](#bookmark78) [[60],](#bookmark79)  [[61],](#bookmark80) [[62],](#bookmark81) [[63],](#bookmark82) [[64],](#bookmark83) [[65],](#bookmark84) [[66]](#bookmark85) |

**3.1 Feature Engineering**

The quality of features, perhaps, is the most important per- spective for the performance of subsequent learning models [[1], [13]](#bookmark1). Such importance is further veriﬁed by the success of deep learning models, which can directly learn a represen- [tation of features from the original data [67]](#bookmark87). 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 applica- tion 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 [8.3.1.](#bookmark88) For now, we focus on feature enhancing methods.

*3.1.1 Feature Enhancing Methods*

In many cases, the original features from the data may not be good enough, e.g., their dimensionality may be too high or samples may not be discriminable in the feature [space [67]](#bookmark87). Consequently, we may want to perform some post-processing on these features to improve the learning performance. Fortunately, while human assistance is still required, there are common methods and principled ways to enhance features. They are listed as follows:

• Dimension reduction: It is the process of reducing the number of random variables under consideration by ob- taining a set of principal variables. Dimension reduction is useful when the features have great redundancy or the feature dimensionality is too high. Techniques of this kind can be divided into feature selection and feature projection. Feature selection tries to select a subset of features from the original ones, where popular methods are greed search and lasso. Feature projection trans- forms original features to a new low-dimensional space, [e.g., PCA [68], LDA [69], and recently developed auto](#bookmark89)- [encoders [70]](#bookmark90).

• Feature generation: Unexplored interactions among origi- nal features, once discovered, may signiﬁcantly improve the learning performance. Feature generation is to con- struct new features from the original ones based on some [pre-deﬁned operations [11], [12], [50], [51], [52], [71],](#bookmark12)

[[72], e.g., multiplication of two features, and standard](#bookmark91) normalization.

• Feature encoding: The last category is feature encoding, which re-interprets original features based on some dic- tionaries learned from the data. Since the dictionary can capture the collaborative representation in the training data, training samples that are not discriminable in the original space become separable in the new space. Popu- [lar examples of this kind are sparse coding [34] (and its](#bookmark35) [convolutional variants [73]) and local-linear coding [74]](#bookmark92). Besides, kernel methods can also be considered as fea- ture coding, where basis functions act as the dictionary. However, kernel methods have to be used with SVM, and basis functions are designed by hand and is not driven by data.

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 en- hancing tools. The ﬁrst one is made up by hyper-parameters of these tools, and conﬁguration exactly refers to these [hyper-parameters [7], [8], [39]](#bookmark8). It covers dimension reduc- tion and feature encoding methods. For example, we need to determine the dimension of features when employing PCA, and the level of sparsity if sparse coding is used.

The second type of search space contains feature to be generated and selected. It commonly considered in feature [generation, e.g., [11], [12], [50], [51], [52], [71], [72]](#bookmark12). Basi- cally, the search space is spanned by operations on original features. One example of new feature generated from plus, minus and times operations is shown in Figure [8.](#bookmark93) For these methods, a conﬁguration is a newly generated feature in the search space.

**3.2 Model Selection**

Once features have been obtained, we need to ﬁnd a model to predict the labels. Models selection contains two com- ponents, i.e., picking up some classiﬁers and setting their corresponding hyper-parameters. In this AutoML setup , the task is to automatically select classiﬁers and set their hyper- parameters so that good learning performance can be obtained.

**new feature**

|  |
| --- |
| **A** X **B - ( C + D )** |



**+**

**operations**



**-**





|  |
| --- |
| **B** |

**original features**

|  |
| --- |
| **A** |

|  |
| --- |
| **C** |

|  |
| --- |
| **D** |

Fig. 8. An example of a newly generated feature (A × B - (C + D)), which is based on plus (+), minus (-) and times ( ×) operations.

*3.2.1 Classiﬁcation Tools*

Many classiﬁcation tools have been proposed in the litera- ture, 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 [[1], [13]](#bookmark1). Some out-of-the-box classiﬁers implemented in scikit-learn are listed in Table [4.](#bookmark95) As can be seen, different hyper-parameters are associated with each classiﬁer. Tra- ditionally, the choice among different classiﬁers and their hyper-parameters are usually determined by human with his/her experience in a trial-and-error manner.

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 |

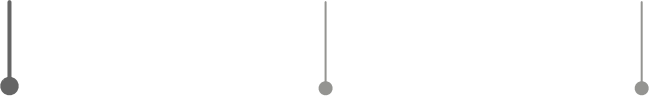
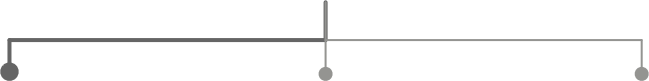
*3.2.2 Search Space*

In the context of model selection, the candidate classiﬁers and their corresponding hyper-parameters make up the search space. Figure [9](#bookmark97) shows a hierarchical structure that [is commonly used to represent the search space [7], [8],](#bookmark8) [[39], [42], [53], [54]](#bookmark53). The rationale behind this structure is that we need to determine the hyper-parameters only if the corresponding classiﬁeris considered.

**3.3 Optimization Algorithm Selection**

The last and the most time consuming step of machine learning is the model training, where optimization is usually involved. For classical learning models, optimization is not a concern, since they usually employs convex loss functions and their performance obtained from various optimization [algorithms are nearly the same [75]](#bookmark99). Hence, efﬁciency is the main focus on the choice of optimization algorithm.

However, as the learning tools get increasingly more complex, e.g. from SVM to deep networks, optimization

8

**hyper-parameters classiﬁers**



|  |
| --- |
| **KNN** |



|  |
| --- |
| **linear SVM** |

|  |
| --- |
| ***c*: class\_weight** |

|  |
| --- |
| ***c*: alpha** |

|  |
| --- |
| ***c*: n\_neighbors** |

|  |
| --- |
| ***c*: C** |

|  |
| --- |
| ***d*: leaf\_size** |

|  |
| --- |
| ***d*: penalty** |

|  |
| --- |
| ***d*: loss** |

|  |
| --- |
| **a conﬁguration** |

|  |
| --- |
| **Bernoulli**  **naive Bayes** |

|  |
| --- |
| ***c*: weights** |

|  |
| --- |
| ***c*: class\_prior** |

|  |
| --- |
| ***d*: binarize** |

|  |
| --- |
| ***d*: ﬁt\_prior** |

Fig. 9. An illustration of the search space for model selection, where KNN, linear SVM and Bernoulli naive Bayes classiﬁers are considered. Hyper-parameters are derived based on Scikit-Learn, “*c:*” indicates that the hyper-parameter is continuous while “*d:*” means that it is discrete. In this ﬁgure, a conﬁguration is made up by the selection of KNN classiﬁer and its values in corresponding hyper-parameters.

is not only the main consumer of computational budgets but also has a great impact on the learning performance [[22], [76]](#bookmark75). Consequently, the goal of algorithm selection is to automatically ﬁnd an optimization algorithm so that efﬁciency and performance can be balanced.

*3.3.1 Optimization Algorithms*

For each learning tool, many algorithms can be used. Some popularly approaches to minimize smooth objective func- tions, like logistic regression, are summarized in Table [5.](#bookmark100) While gradient descent (GD) does not involve extra param- eters, it suffers from slow convergence and expensive per- iteration complexity. Two popular variants of GD are limited memory-BFGS (L-BFGS) and stochastic gradient descent (SGD). The former is more expensive but converges faster [[77], while in the latter each iteration is very cheap but many](#bookmark101) [iterations are need before convergence [75]](#bookmark99).

TABLE 5

Some popular optimization algorithms 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. η0 /(1+λη0t)c where tis the number of iterations, η0 , λ

and c are continuous hyper-parameters [[75])](#bookmark99).

|  |  |  |  |
| --- | --- | --- | --- |
|  | number of hyper-parameters | | |
|  | total | discrete | continuous |
| GD | 0 | 0 | 0 |
| L-BFGS | 1 | 1 | 0 |
| SGD | 4 | 1 | 3 |

*3.3.2 Search Space*

Traditionally, both the choices of optimization algorithms and their hyper-parameters are made by humans based on their understanding of the learning tools and observations of the training data. To automate algorithm selection, the search space is determined by conﬁgurations of optimiza- tion algorithms, which contains the choice of optimization algorithms and the values of their hyper-parameters, e.g, [[55], [56], [57], [58]](#bookmark70). There is also naturally a hierarchy in such search space, which is similar to that shown in Figure [9,](#bookmark97)

as hyper-parameters of an algorithm will be considered only when the corresponding algorithm is selected.

**3.4 Full Scope**

In the this section, we discuss the full pipeline in Figure [1.](#bookmark7) There are generally two classes of full-scope AutoML ap- proaches.

• The ﬁrst one is general case. The learning process consid- ered in this case is a combination of feature engineering, model selection and algorithm selection. The resulting search space is also an union of previous ones discussed in Section [3.1-3.3, as has been considered in [7], [8], [39],](#bookmark86)

[[42], [78] already.](#bookmark66)

• The second one is NAS, which targets at searching good deep network architectures that suit the learning prob- lem. There are three main reasons why we discuss it in parallel with the full scope. First, NAS itself is currently an extremely hot research topic under which many pa- [pers have been published, i.e., [9], [22], [25], [26], [45],](#bookmark10)

[[59], [60], [61], [62], [63], [64], [65], [66], [79] and etc](#bookmark78). The second reason is that the application domain for deep networks is relative clear, i.e., the domain of learn- ing from low-semantic-level data such as image pixels. Finally, since the application domain is clear, domain- speciﬁc network architectures can fulﬁll the learning purpose, where feature engineering and modelselection are both done by NAS.

*3.4.1 Network Architecture Search (NAS)*

Before describing the search space of NAS, let us look at what is a typical architecture of a convolutional neural network (CNN). As shown in Figure [10, basically, CNN is](#bookmark102) mainly made up by two parts,i.e., a series of convolutional layers and a fully connected layer in the last.

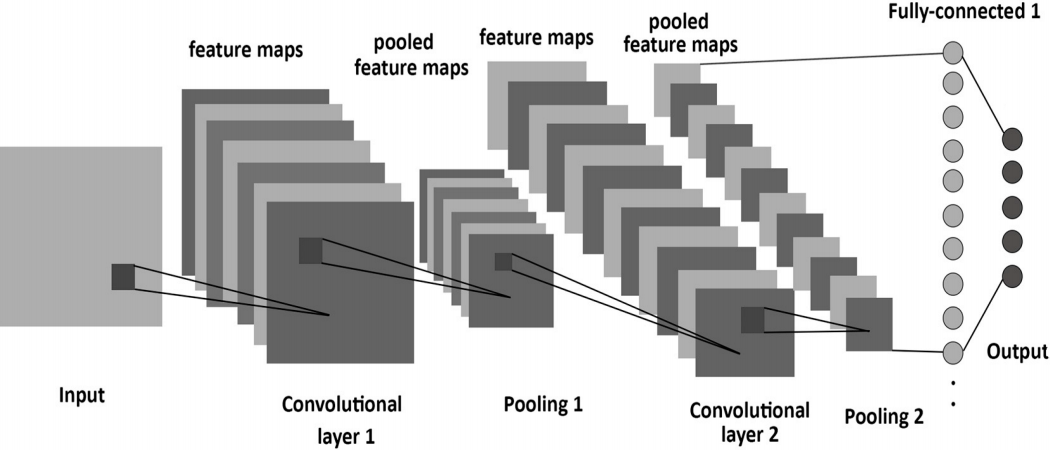


Fig. 10. A very typical CNN architecture, which contains ﬁlters, pooling and jump connections (the image is from [[80])](#bookmark103).

The performance of a CNN is mostly inﬂuenced by the [design of convolutional layers [81], of which some common](#bookmark104) design choices are listed in Figure [11.](#bookmark105) The search space is made up by above design choices among all convolutional layers, and one conﬁguration for NAS is a point in such a search space.

Among various DNN architectures, we focus on CNN in this survey, but the presented idea can be similarly applied [for other architectures, such as long-short-term-memory [82]](#bookmark106) [and deep sparse networks [83]](#bookmark107).

9

• stride height

• number of ﬁlters

• ﬁlter height

• ﬁlter width

• stride width

• skip connections

Fig. 11. Some common design choices for one convolutional layer in a CNN.

**4 BASIC TECHNIQUES FOR OPTIMIZER**

Once the search space is deﬁned, as in the proposed frame- work (Figure [6), we need to ﬁnd an optimizer to guide the](#bookmark52) 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 tech- niques can be used for the optimizer, and the last one clariﬁes the efﬁciency of techniques. While efﬁciency is a major concern in AutoML (see Remark [2.1), in this section,](#bookmark46) we do not categorize existing techniques based on it. This is because the search space is so complex where convergence rates for each technique are hard to analyze. We take it as one future direction in Section [8.3.4.](#bookmark109)

At the same time, experienced techniques (Section [6) can](#bookmark59) 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.](#bookmark110)

**4.1 Simple Search Approaches**

Simple search is a naive search approach, they make no assumptions about the search space. Each conﬁguration in the search space can be evaluated independently. Grid search and random search are two common approaches.

• Grid search (brute-force): it is the most traditional way of hyper-parameters tuning. To get the optimal hyper- parameter setting, grid search have to enumerate every possible conﬁgurations in the search space. Discretization is necessary when the search space is continuous.

• Random search: it randomly samples conﬁgurations in the search space. Random search empirically performs [better than brute-force grid search [47]](#bookmark57). As shown in Figure [12, random search can explore more on important](#bookmark112) dimensions than grid search.

Simple search approaches gather the feedbacks from the evaluator merely to keep track of the good conﬁgurations. Because simple search does not exploit the knowledge gained from the past evaluations, it is usually inefﬁcient. However, due to its simplicity, it is still popularly used in AutoML.

**4.2 Optimization from Samples**

[Optimization from samples [84] is a kind of smarter search](#bookmark114) approach compared with simple ones in Section [4.1.](#bookmark111) It it- eratively generates new conﬁgurations based on previously

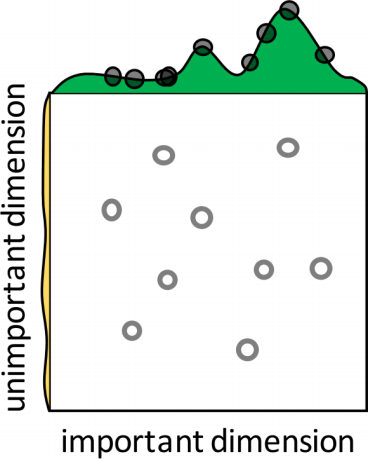
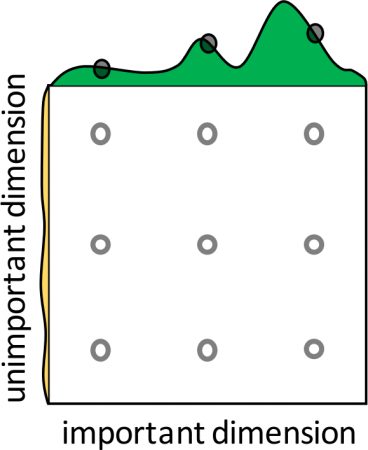
10

TABLE 6

Comparison of various techniques for the optimizer based on Remark [4.1.](#bookmark108) Multi-step means the conﬁguration can be made up by a several

decision steps.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| type | method | conti- nuous | discrete | multi- step | example | feedback in example |
| simple search | | √ | √ | × | [[47]](#bookmark57) | none |
| optimization from samples | evolutionary algorithm | √ | √ | × | [[15]](#bookmark115) | performance on validation set |
| Bayesian optimization | √ | √ | × | [[39]](#bookmark53) | performance on validation set |
| reinforcement learning | √ | √ | √ | [[9]](#bookmark10) | performance on validation set (reward) and a sequence of conﬁgurations (state) |
| gradient descent | | √ | × | × | [[41]](#bookmark116) | performance on validation set and gradients w.r.t hyper-parameters |
| greedy search | | × | √ | √ | [[10]](#bookmark11) | performance on validation set |



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

Fig. 12. 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 [[47])](#bookmark57).

evaluated samples. Thus, it is also generally more efﬁcient than simple search methods. Besides, it does not make speciﬁc assumptions about the objective.

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

*4.2.1 Heuristic Search*

Heuristic search methods are often inspired by biologic behaviors and phenomenons. They are widely used to solve optimization problems that are non-convex, non-smooth, or even non-continuous. The majority of them are population- based optimization methods, and differences among them are how to generate and select populations. The framework of heuristic search is shown in Figure [13.](#bookmark117) The initialization step generates the ﬁrst population (a bunch of conﬁgura- tions in AutoML). At each iteration, a new population is generated based on the last one, and the ﬁtness (perfor- mances) of the individuals are evaluated. The core idea of heuristic search is how to update the population.

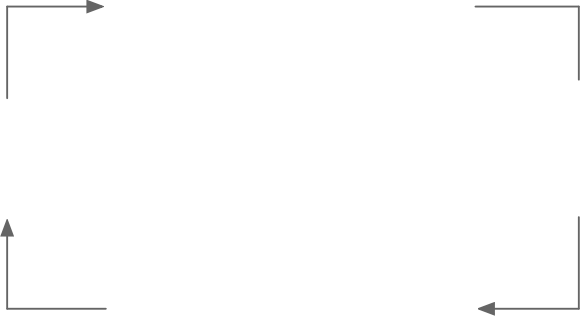
Some popular heuristic search methods are listed as follow:

• Particle swarm optimization [(PSO) [53]: PSO is inspired](#bookmark67) by the behavior of biological communities that exhibit both individual and social behavior; examples of these communities are ﬂocks of birds, schools of ﬁshes and swarms of bees. Members of such societies share common goals (e.g., ﬁnding food) that are realized by exploring its environment while interacting among them. At each iteration, the population is updated by moving towards

the best individuals. PSO optimizes by searching the neighborhoods of the best samples. It has a few hyper- parameters itself and can be easily parallelized. In such way, PSO hopes to ﬁnd the best position in search space. • Evolutionary algorithms [[85]: Evolutionary algorithms are](#bookmark119) inspired by biological evolution. The generation step of evolutionary algorithms contains crossover and muta- tion. Crossover involves two different individuals (an- cestors) from the last generation. It combines them in some way to generate an new individual. In principal, the more promising an individual is, the more likely is it to be chosen as an ancestor. Mutation, on the other hand, slightly changes an individual to generate a new one.

With crossover mainly to exploit and mutation mainly to explore, the population is expected to evolve towards better performance.

**initialization**



|  |
| --- |
| **evaluator**  evaluate new population |

|  |
| --- |
| **optimizer**  generate new population |

|  |
| --- |
| **population**  evaluated conﬁgurations |

|  |
| --- |
| **new population**  new conﬁgurations |

Fig. 13. Work ﬂow of heuristic search. It is the population-based search approach, and starts with a initialization process.

The above methods have been widely applied in Au- toML. For example, evolutionary algorithms has been ap- [plied in feature selection and generation [50], [86], [87],](#bookmark63) [[88], [89], and model selection [90]](#bookmark120). PSO has been used [for model selection [53], [91], feature selection for support](#bookmark67) [vector machine (SVM) [91], and hyper-parameter tuning](#bookmark121) [for deep networks [92]](#bookmark122). While evolutionary algorithms have [already been used in NAS one decade ago [93], [94], [95], it is](#bookmark123) only recently that better performance than human designed [architecture are achieved [15], [96], [97], [98]. In these works,](#bookmark115) network structures are encoded with binary strings, on which the evolutionary operations are performed.

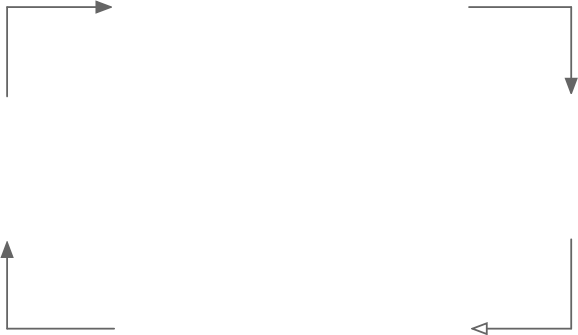
*4.2.2 Model-Based Derivative-Free Optimization*

The general framework of model-based derivative-free op- timization is showed in Figure [14.](#bookmark124) It is different from the

11

heuristic search in sense that model-based optimization builds a model based on visited samples. Full utilization of feedbacks from the evaluator helps it generate more promising new samples. The popular methods of this kind are Bayesian optimization, classiﬁcation-based optimization and optimistic optimization:

**initialization**



|  |
| --- |
| **model-based**  **optimizer**  generate new conﬁguration |

|  |
| --- |
| **samples**  evaluated conﬁgurations |

|  |
| --- |
| **evaluator**  evaluate new conﬁguration |

|  |
| --- |
| **new samples**  new conﬁgurations |

Fig. 14. 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 built on previous samples.

• Bayesian optimization [[99], [100], [101]: Bayesian optimiza](#bookmark125)- tion builds a probabilistic model, e.g., Gaussian pro- [cess [102], [103], tree-based model [104], [105], or deep](#bookmark126) [network [106], that maps the conﬁgurations to their](#bookmark127) performance with uncertainty. Then, it deﬁnes an ac- quisition function based on the probabilistic model, e.g., expected improvement, upper conﬁdence bounds, to bal- ance exploration and exploitation during search. At each iteration, a new sample is generated by optimizing the acquisition function, and used to update the probabilistic model once it is evaluated.

• Classiﬁcation-based optimization [(CBO) [44], [107], [108]:](#bookmark129) Based on previous samples, classiﬁcation-based opti- mization learns a classiﬁer that divides the search space into positive and negative areas. Then, new samples are randomly generated in the positive area where it is more likely to get better conﬁgurations. The learned classiﬁers can be very simple, which produce decision boundary in parallel with coordinates of the search space. Thus, classiﬁcation-based optimization is usually very efﬁcient.

• Simultaneous Optimistic optimization [(SOO) [109], [110]:](#bookmark130) SOO is a branch-and-bound optimization algorithm. A tree structure is built on the search space where each leaf node bounds a sub area. SOO deeply explores the search space by expanding leaf nodes according to some strate- gies. Trough the tree model, SOO can balance exploration and exploitation to ﬁnd the global optimum when the ob- [jective function is local-Lipschitz continuous [110]](#bookmark131). But it also suffers from the curse of dimensionality because the tree grows extremely complicated when dimensionality the search space is high.

Due to its long history and sound theoretical justiﬁ- cation, Bayesian optimization is perhaps the most popu- larly used method in this category. Early attempts include [[99], [102], [111], [112], which have shown a promising](#bookmark125) performance of Bayesian optimization for hyper-parameter tuning. [Later on, it has been applied in sklearn [7], [113]](#bookmark8)

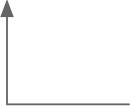
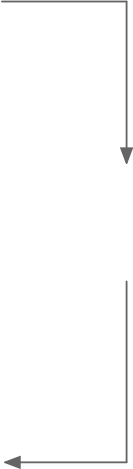
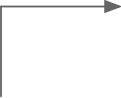
[and Weka [8], [39] for automatic conﬁguration of out-of](#bookmark9)- box classiﬁers. More recently, CBO has been developed as a better method than Bayesian optimization for hyper- [parameter tuning [44] and policy search [107]](#bookmark129).

*4.2.3 Reinforcement Learning*

[Reinforcement learning (RL) [114] is a very general and](#bookmark132) strong optimization framework, which can solve problems with delayed feedbacks. Figure [15](#bookmark128) illustrates its general framework when used in AutoML. Basically, the policy in RL acts as the optimizer, 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.

**current state**

uncompleted conﬁguration



**feedbacks**

|  |
| --- |
| **action**  candidate decisions |

|  |
| --- |
| **policy**  **optimizer**: generate candidate decisions |

|  |  |  |
| --- | --- | --- |
| |  | | --- | | **reward**  evaluation results |  |  | | --- | | **new state**  select new decision based on evaluations | |

|  |
| --- |
| **environment**  **evaluator**: estimate the  delayed reward and  update state |

Fig. 15. Workﬂow of reinforcement learning. It is different from heuristic search and model-based derivative-free optimization as feedbacks need not be immediately returned for receiving a conﬁguration.

Resulting from the above-mentioned unique property, [RL is recently used in NAS [9], [16]](#bookmark10). 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 optimizer. However, the performance of an architecture can only be evaluated after its whole structure is composed, which implies a delayed reward. Thus, the iterative architecture generation naturally follows the property of RL (see details in Sec- tion [7.2)](#bookmark133). However, due to the delayed feedbacks, AutoML with reinforcement learning is highly source-consuming, and more efﬁcient methods needs to be explored. Some current endeavors addressing this problems are learning [transferable architectures from smaller data sets [23], and](#bookmark74) [cutting the search space by sharing parameter [115], [116]](#bookmark134).

Besides, a special case of RL,i.e., bandit-based approach, where rewards are returned for each action without a delay, is introduced to AutoML for hyper-parameter optimization [[117], [118]](#bookmark135). Finally, RL has also been used for optimization [algorithms search [22], automated feature selection [119],](#bookmark75) [and training data selection in active learning [120]](#bookmark136).

**4.3 Gradient descent**

Optimization problems of AutoML is very complex, and the objective is usually not differentiable or even not continu- ous. Thus, gradient descent is not as popular as methods in

12

Section [4.2.](#bookmark113) However, focusing on some differentiable loss [function [121], e.g., squared loss and logistic loss, continu](#bookmark137)- ous hyper-parameters can be optimized by gradient descent. Compared with above methods, gradients offer the most accurate information where better conﬁgurations locates.

Unlike traditional optimization problems whose gradi- ents can be explicitly derived from the objective, in AutoML problems, the gradients need to be numerically computed. Usually, this can be done with ﬁnite differentiation meth- [ods [121] but at high costs](#bookmark137). For some traditional machine learning methods, e.g., logistic regression and SVM, the ap- proximate gradient is proposed to search continuous hyper- [parameters [122]](#bookmark138). The computation of exact gradients relies on the convergence of model training. Through inexact gra- dient, hyper-parameters can be updated before the model training converges, which makes gradient descent method more efﬁcient.

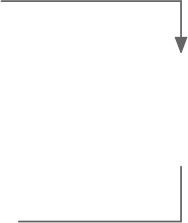
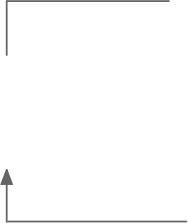
Another way to compute gradients is through reversible [learning (also named as automatic differentiation) [48]](#bookmark139). It computes gradients with chain-rule, which is also used in the back-propagation process of network training. It has [been applied in deep learning hyper-parameter search [41],](#bookmark116) [[123]](#bookmark140).

**4.4 Greedy search**

Greedy search is a natural strategy to solve multi-step decision-making problem. It follows a heuristic that makes locally optimal decision at each step with the intent of ﬁnding a global optimum. For example, in travel salesman problem, greedy search selects to visit the nearest city at each step of the journey. Greedy search cannot ﬁnd the global optimum, but it can usually ﬁnd a local optimum which approximates the global optimum in a reasonable time cost. Besides, such good empirical performance is also theoretically justiﬁed in many applications, e.g., feature [selection [124] and submodular optimization [125]](#bookmark141).

**past decisions**

uncompleted conﬁguration



|  |
| --- |
| **new decision**  selected according to evaluation results |



|  |  |
| --- | --- |
| **evaluator**  evaluate candidate decisions |  |

|  |
| --- |
| **optimizer**  generate candidate decisions |

|  |
| --- |
| **candidate decisions** |

Fig. 16. Workﬂow of greedy search. It targets at multi-step decision problems, where a local optimal decision is made at each step.

Multi-step decision-making problems are also com- monly encountered in AutoML. For example, in NAS prob- lem, the architecture for each layer needs to be decided, [and greedy search is applied in [24] for multi-attribute](#bookmark143) [learning problems; greedy search is also employed in [10],](#bookmark11) [[116] to search block structures within a cell, which is later](#bookmark144) used to construct a full CNN. Besides, in feature generation problems where the search space can be prohibitively large, [greedy search is recently considered in [11], [12] to generate](#bookmark12) more discriminative features with original ones.

**4.5 Other techniques**

Finally, there are some techniques that do not fall into above categories. They are usually developed case-by-case. Currently, a popular one is to change the landscape of the search space so that more powerful optimization techniques can be used. For example, in NAS, as the conﬁguration space [is discrete soft-max is used in [46] to change the search space](#bookmark145) to a continuous one, which enables the usage of gradient descent instead of RL; an encoder-decoder framework is [used in [126], which also maps discrete conﬁgurations into](#bookmark146) a continuous search space.

**5 BASIC TECHNIQUES FOR EVALUATOR**

In Section [4, we discussed how to choose a proper basic](#bookmark30) technique for the optimizer. In this section, we will visit techniques for another component, i.e., the evaluator in Figure [6.](#bookmark52) Once a candidate conﬁguration is generated, the evaluator needs to measure its performance. This process is usually very time-consuming as it involves model training for most of the times.

**Remark 5.1.** Three important questions to determine the basic technique for the evaluator are:

(A). Can the technique provide fast evaluation?

(B). Can the technique provide accurate evaluation?

(C). What feedbacks need to be provided by the evaluator?

As illustrated in Figure [17, there is usually a trade-off be](#bookmark142)-

tween the focus of questions (A) and (B) as faster evaluation usually leads to degraded result, i.e., with lower accuracy but larger variance. The last question in Remark [5.1](#bookmark147) is a de- sign choice,it also depends on choices of the optimizer. For example, as shown in Table [6, while Bayesian optimization](#bookmark110) only requires the performance, gradient descent methods in addition need gradient information.

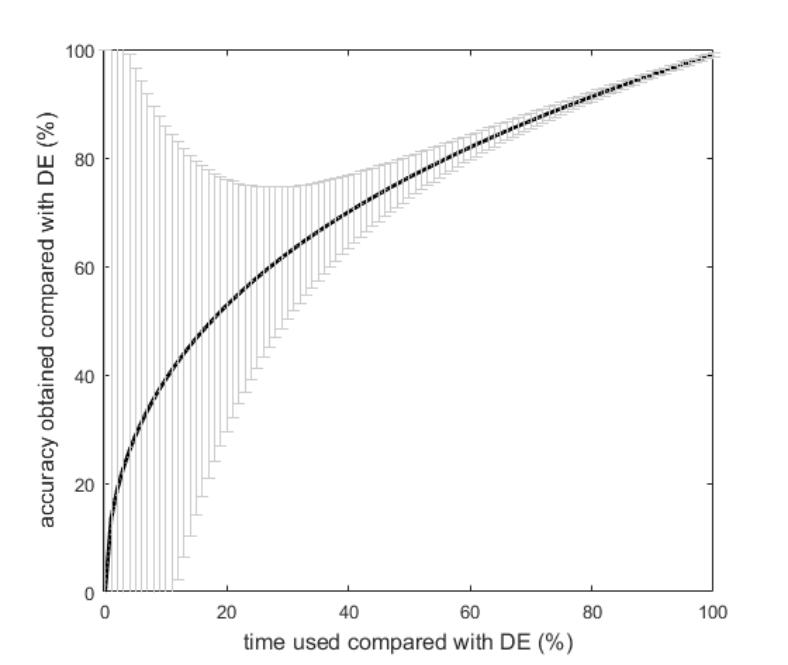


Fig. 17. The tradeoff between evaluation’s accuracy and time, where DE denotes direct evaluation (see Section [5.1) and both time and accuracy](#bookmark148) are measured relatively to that of DE. The gray lines indicate variance in accuracy obtained.

**5.1 Techniques**

Unlike the optimizer, the evaluator seldom cares about the search space of conﬁgurations. Since the majority of opti- mizers generate candidates that can be directly applied on

13

learning tools, the most simple and straightforward way to evaluate themisto learn the model parameters and estimate the performance:

• Direct evaluation: This is the simplest method, where the model parameters are learned on the training set, and the performance is measured on the validation set afterwards. Direct evaluation is often accurate but expen- sive.

In AutoML problems, usually, many candidate conﬁgu- rations will be generated and evaluated. The direct evalua- tion approach, though very accurate, is usually prohibitively expensive to be invoked repeatedly. Consequently, some other methods have been proposed for acceleration by trad- ing evaluation accuracy for efﬁciency:

• Sub-sampling: As the training time depends heavily on the amount of training data, an intuitive method to accelerate evaluation is to train parameters with a subset of the training data. This can be done by either using a subset of samples, a subset of features or multi-ﬁdelity evaluations

[[127]](#bookmark149). In general,the less training data is used, the faster and more noisy will be the evaluation.

• Early stop: In classical machine learning, early stop is a popular method to prevent over-ﬁtting. However, in the context of AutoML, it is usually used to cut down the training time for unpromising conﬁgurations. Such conﬁgurations can usually be easily identiﬁed at the early stage of model training, with their performance [monitored on the validation set [65], [105], [128], [129]](#bookmark84). If a poor early-stage performance is observed, the evaluator can terminate the training and report a low performance to indicate that the candidate is unpromising. Early stop cuts down the total running time of AutoML, but also introduces noise and bias to the estimation as some conﬁgurations with bad early-stage performance may eventually turnout to be good after sufﬁcient training.

• Parameter reusing: Another technique is to use parame- ters, of the trained models in previous evaluations, to warm-start the model training for the current evaluation. Intuitively, parameters learned with similar conﬁgura- tions can be close with each other. For a candidate that is close to previously evaluated conﬁgurations, parameters of the latter can be a good start point for training and may lead to faster convergence and better performance. In [such cases, parameter reusing can be very helpful [130]](#bookmark150). However, as different start points may lead convergence to different local optima, it sometimes brings bias in the [evaluation [131]](#bookmark151). Parameter reusing can be considered as one of the most straightforward applications of transfer learning, which will be further discussed in Section [6.2.](#bookmark48)

• 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 [10], [43], [59], [129], [132], [133]](#bookmark11). These models, serving as surrogate evaluators, spare the com- putationally expensive model training, and signiﬁcantly accelerate AutoML. Surrogate evaluators can predict not only the performance of learning tools, but also the train- ing time and model parameters. However, their appli-

cation scope is limited to hyper-parameter optimization since other kinds of conﬁgurations are often hard to quantize,which hinders surrogate model training. In Sec- tion [6.1, we will introduce meta-learning techniques that](#bookmark47) are promising to address this problem. Finally, it should be noted that, while surrogate models are also used in sampled-based optimization techniques (Section [4.2.2),](#bookmark118) they do not act as surrogate evaluators, but are used to generate potentially promising conﬁgurations.

Direct evaluation, due to its simplicity and reliability, is perhaps the most commonly used basic evaluator technique in AutoML. Sub-sampling, early stop, parameter reusing, and surrogate evaluator, enhance Direct evaluation in various directions, and they can be combined for faster and more accurate evaluation. However, the effectiveness and efﬁ- ciency of these techniques depend on the AutoML problem and data, and it is hard to quantitatively analyze their improvement over Direct evaluation.

While the basic techniques for evaluators are seemingly much fewer than those for optimizers, it does not make the evaluator less important. In fact, the accuracy and efﬁciency of the evaluator have a great impact on both the quality of the ﬁnal result and the runtime of the AutoML program. In order to make evaluators more powerful, various techniques based on, e.g., meta-learning and transfer learning have been introduced to AutoML. They will be discussed in Section [6.](#bookmark59)

**6 EXPERIENCED TECHNIQUES**

In Section [2.2, we discussed the general framework to](#bookmark37) automatically construct learning tools for given learning problems. The framework highlights a search procedure that comprises conﬁguration generation and evaluation. In this section, we review experienced 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.

**Remark 6.1.** It should be noted that the scope of meta-learning overlaps with that of transfer learning since they all aim to exploit experience gained from past learning practices. Besides, there are no uniﬁed deﬁnitions of meta-learning and some researchers also [take transfer learning as a special case of meta-learning [29], [49],](#bookmark22) [[134]](#bookmark152). The discussion on the intrinsic similarities and differences between meta-learning and transfer learning is beyond the scope of this survey. Here, we distinguish them with the following criterion: an approach is a meta-learning one if it extracts meta- knowledge about learning problems and tools (e.g., meta-features and performance) from past learning and trains a meta-learner to facilitate the future; otherwise, it is a transfer learning approach if it directly uses the ﬁnal or intermediate results (e.g., best con- ﬁguration, or surrogate model) of past learning. Please compare Figure [18](#bookmark153) and [19.](#bookmark154)

**6.1 Meta-Learning**

Meta-learning in general learnshow speciﬁclearning tools perform on given problem from past experiences, with the

14

TABLE 7

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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| application | | meta-knowledge  (extracted from past experience) | | | meta-learner  (applied on upcoming problems) | | examples |
| learning problem | learning tool | other | input | output |
| conﬁguration evaluation | 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 | [[57],](#bookmark72) [[135],](#bookmark156)  [[136]](#bookmark157) |
| general  conﬁguration  evaluation | meta-  features of  data  (optional) | conﬁgura-  tions, or  meta-features  of conﬁgura-  tions  (optional) | performance of conﬁgurations | meta-features of  data, and  conﬁgurations  or meta-features  of  conﬁgurations | performance  of conﬁgura-  tions | [[11],](#bookmark12) [[137]](#bookmark158) |
| conﬁguration generation | promising  conﬁguration  generation | meta-  features of  data | - | well-performing conﬁgurations or generation strategy | meta-features of data | promising  conﬁgura-  tions or  generation  strategy | [[138],](#bookmark159)  [[139],](#bookmark160) [[140]](#bookmark161) |
| warm-  starting  conﬁguration  generation | meta-  features of  data | - | well-performing conﬁgurations | meta-features of data | promising  initial conﬁg-  urations | [[141],](#bookmark162)  [[142],](#bookmark163) [[143]](#bookmark164) |
| search space reﬁning | - | conﬁgura- tions | importance of  conﬁgurations, or  promising search  regions | conﬁgurations of learning tools | reﬁned search space | [[144],](#bookmark165)  [[145],](#bookmark166) [[146]](#bookmark167) |
| 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 | [[149],](#bookmark169)  [[150],](#bookmark170) [[151]](#bookmark171) |

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 evaluation (for the evaluator); 2) meta- learning for conﬁguration generation (for the 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)](#bookmark27). It is, however, signiﬁcantly different from classical machine learning since it aims at totally different tasks and, consequently, learns from different experiences. Table [8](#bookmark172) 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 [18](#bookmark153) illustrates the general framework of meta-learning. First, learning problems and tools are characterized. Such characteristics (e.g., statistical properties of the data set, hyper-parameters of learning tools) are often named meta-features, as thoroughly re-

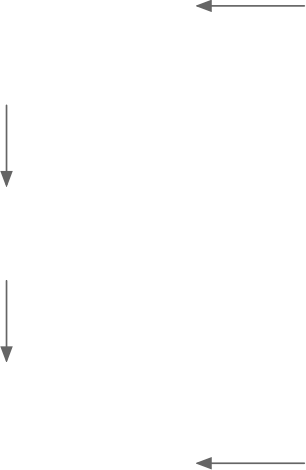
TABLE 8

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 | |

[viewed in [28], [29], [135]](#bookmark21). Then, meta-knowledge is ex- tracted from past experiences. In addition to meta-features, empirical knowledge about the goal of meta-learning, such as performance of learning tools or 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 statisti- cal methods, can serve to generate the meta-learners. The trained meta-learner can be applied on upcoming, charac- terized learning problems to make predictions of interest.

Meta-learning helps AutoML, on the one hand, by char- [acterizing learning problems and tools [29], [152]](#bookmark22). Such char- acteristics can reveal important information about the prob- lems and tools, for example, whether there are concept drift [in the data [150], [151], or whether a model is compatible](#bookmark170) [for particular machine learning tasks [153], [154]](#bookmark173). Further- more, with these characteristic, similarities among different tasks and tools can be evaluated, which enables knowledge

**meta-knowledge past learning experience**

**characterize characterize**

|  |  |  |
| --- | --- | --- |
| |  | | --- | | **learning problems** |  |  | | --- | | **learning tools** | |

**train**

|  |  |  |
| --- | --- | --- |
| |  | | --- | | **meta-features** |  |  | | --- | | **other** | |

|  |
| --- |
| **meta-learner** |

**upcoming learning problem**

**apply**

|  |  |  |
| --- | --- | --- |
| |  | | --- | | **learning problems** |  |  | | --- | | **learning tools** | |

|  |  |  |
| --- | --- | --- |
| |  | | --- | | **meta-features** |  |  | | --- | | **other** | |

**characteristics of**

**upcoming problems**

Fig. 18. The general framework of meta-learning techniques in AutoML.

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 [141],](#bookmark162) [[142], [143], [155]](#bookmark163). 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 com- [putational expensive training and evaluation of models [11],](#bookmark12) [[57], [135], [136], [137]](#bookmark72). They can also generate promising conﬁgurations, which can directly specify a learning tool or [serve as good initialization of the search [141], [142], [143], or](#bookmark162) [suggest effective search strategies [52]. Hence, meta-learning](#bookmark65) 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 corre- sponding meta-knowledge and meta-learners, as noted in Remark [6.2.](#bookmark174)

**Remark 6.2.** 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 pur- pose?

Table [7](#bookmark155) summarizes the meta-knowledge and meta- learners that should be extracted and trained for different purposes, according to existing works in the literature.

*6.1.2 Conﬁguration Evaluation*

The most computation-intensive step in AutoML is conﬁg- uration evaluation, due to the cost of model training and validation. Meta-learners can be trained as surrogate evalu- ators 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 learn- ing problem, whether or how a class of machine learn- ing models is applicable so that the most suitable and

15

promising conﬁguration can be identiﬁed. The meta- knowledge includes the meta-features of learning prob- lems 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 per- [formance [55], [154], [156], applicability [153], [157]2,](#bookmark70) or [ranking [158], [159], [160], [161] of models](#bookmark176). More recent [research on this topic include active testing [136], [162],](#bookmark157) [runtime prediction [57], [163], and more sophisticated](#bookmark72) [measurements for models [164], [165]](#bookmark177). A more complete [review of on this topic can be found in [135]](#bookmark156).

• General conﬁguration evaluation: The evaluation for other kinds of conﬁgurations can equip meta-learning in simi- [lar ways: in ExploreKit [11], ranking classiﬁers are trained](#bookmark12) [to rank candidate features; in [137], meta-regressor is](#bookmark158) trained to score kernel widths as hyper-parameters for SVM.

In short, with the purpose to accelerate conﬁguration evaluation, meta-learners are trained to predict the perfor- mance 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 [135],](#bookmark156) [[149], where all possible choices have been enumerated,](#bookmark169) best conﬁgurations can be directly selected according to the scores and rankings predicted by the meta-learner.

*6.1.3 Conﬁguration Generation*

Meta-learning can also facilitate conﬁguration generation by learning, e.g., conﬁgurations for speciﬁc learning problems, 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 di- rectly generate well-performing conﬁgurations forgiven learning problem. For this purpose, meta-knowledge indicating the empirically good conﬁgurations are ex- tracted, and the meta-learner take the characteristics of learning problem as input and predict promising conﬁg- [urations, such as kernel [138], adaptive network archi](#bookmark159)- [tectures [139], [140]](#bookmark160). Additionally, it is also possible to learn promising conﬁguration generation strategies. For [example, in [52], the authors trained a meta-learner that](#bookmark65) predicts useful feature transformations.

• 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 iden- tify the past tasks that are closest to it in the meta- feature space, and use their best-performing conﬁgura- tions to initialize search. Most work of this kind focus on hyper-parameter tuning, with particle swarm opti- [mization [141], [155], evolutionary algorithm [142], or](#bookmark162) [Bayesian optimization [7], [143], [166], [167]](#bookmark8).

• Search space reﬁning: Meta-learning can accelerate con- ﬁgration search by reﬁning the search space. Existing

2. Literally, applicability indicates whether a machine learning model is useful (to perform well) for a given problem [[153]](#bookmark173). Better performance means higher applicability.

16

works of this line make effort to evaluate the importance [of conﬁgurations [144], [146], [168], or identify promising](#bookmark165) [regions in the search space [145]](#bookmark166).

*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 Au- toML. 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 automate this procedure by detecting concept drift and dynamically adapting learning tools to it:

• Concept drift detection: With statistics of data or features, we can detect if concept drift present in a learning prob- lem. [In [147], attributes that might provide contextual](#bookmark179) clues, which indicate the changes in concept, are iden- tiﬁed based on meta-learning. [In [148], a non-parametric](#bookmark168) approach is proposed to detect concept drift. A new class of distance measures is 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 [149], [150], [151]](#bookmark169). Such approaches are similar to those in promising conﬁguration generation.

*6.1.5 Summary*

We have thus far reviewed major meta-learning techniques in the context of AutoML. Meta-learning improves the per- formance of AutoML by extracting useful knowledge from past experiences. However, applying meta-learning requires certain efforts, as will be discussed in Section [8.3.2.](#bookmark180)

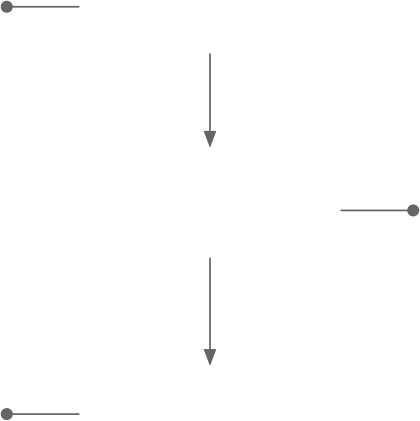
**6.2 Transfer Learning**

[Transfer learning, according to the deﬁnition in [31], tries](#bookmark24) 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 con- ﬁguration 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 is often the ﬁnal or intermediate results of the source conﬁguration generation or evaluation, such as the [conﬁgurations or parameters of the learning tool [23], [64],](#bookmark74) [[81], [115], or the surrogate model trained during hyper](#bookmark104)- [parameter training [169], [170], [171]](#bookmark181). Figure [19](#bookmark154) illustrates how transfer learning works in AutoML. Remark [6.3](#bookmark182)points out the key issues in applying transfer learning, and Ta- ble [6.2](#bookmark48)summarizes the different source, target, and transfer- able knowledge involved in transfer learning in the existing AutoML literature.

**Remark 6.3.** 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?

|  |
| --- |
| **source** |

|  |  |
| --- | --- |
| |  | | --- | | **learned models** |   **search**  **strategies**  **…** |

**extract**

|  |
| --- |
| **learning tool conﬁgurations** |

|  |  |  |
| --- | --- | --- |
| |  | | --- | | **conﬁguration generation** |  |  | | --- | | **conﬁguration generation** | |

**apply**

|  |
| --- |
| **target** |

|  |
| --- |
| **conﬁguration generation** |



|  |
| --- |
| **transferable knowledge** |

|  |  |
| --- | --- |
| |  | | --- | | **conﬁguration generation** | |

Fig. 19. 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 (for the optimizer), and 2) conﬁguration evaluation (for the evaluator).

*6.2.1 Conﬁguration Generation*

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

• Surrogate model transfer: Bayesian optimization for hyper- parameters suffers from the cold-start problem, as it is ex- pensive 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, by transferring the surrogate model [169],](#bookmark181)

[[170] or its components such as kernel function [171]](#bookmark183). A more generalized case is hyper-parameter optimization [machine [173] where a transfer function incorporates the](#bookmark184) loss function of previously visited learning tasks.

• Network cell transfer: Transfer learning is especially widely-used in NAS due to the transferability of net- works. [In [23], [64], the NAS problem is converted to](#bookmark74) searching for architecture building components, often named as ‘cells’, that can be learned with low costs on small datasets 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 [112], Bayesian optimization](#bookmark185) is accompanied with multi-task Gaussian process models so that knowledge gained from past tuning tasks can be transfered to warm-start search. [In [174], a multi-task neural](#bookmark186) AutoML controller is trained to learn hyper-parameters for neural networks.

17

TABLE 9

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| applications | | source | target | knowledge | example |
| conﬁguration generation | surrogate model transfer | past hyper-parameter optimization | current hyper-parameter optimization | surrogate model, or model components | [[169],](#bookmark181) [[170],](#bookmark183)  [[171]](#bookmark187) |
| network cell transfer | past network  architecture search | current network architecture search | network cells | [[23],](#bookmark74) [[64]](#bookmark83) |
| conﬁguration evaluation | model parameter transfer | past architecture evaluation | current architecture evaluation | model parameter | [[81],](#bookmark104) [[115]](#bookmark134) |
| function-preserving transformation | past architecture evaluation | current architecture evaluation | the function represented by the network | [[25],](#bookmark76) [[26],](#bookmark188)  [66], [172] |

*6.2.2 Conﬁguration Evaluation*

In the search for promising learning tools, a great num- ber 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 transfer: The most straightforward method is to transfer parameters from trained architec- [tures to initialize new ones. According to [81], initializing](#bookmark104) network with transferred features layers, followed by ﬁne-tuning, brings improvement in deep neural network performance. [Following this idea, in [115], child networks](#bookmark134) are forced to share weights so that the training costs can be signiﬁcantly reduced.

• Function-preserving transformation: Another line of re- search focus on the function-preserving transformation, [ﬁrst proposed in Net2Net [172] where new networks are](#bookmark190) initialized to represent the same functionality of a given trained model. This approach has been proven capable to signiﬁcantly accelerate the training of new network ar- [chitectures [25]](#bookmark76). Additionally, function-preserving trans- formation also inspires new strategies to explore the [network architecture space in recent approaches [26],](#bookmark188) [[66]](#bookmark85).

*6.2.3 Summary*

As we can observe, the applications of transfer learning in AutoML is relatively limited. Most approaches focused on the network architecture search problem, and the transfer- ability of knowledge is not well addressed in an automatic manner, which motivates the discussion in Section [8.3.2.](#bookmark180)

**7 REPRESENTATIVE EXAMPLES**

In this section, we revisit three examples mentioned in Section [1, i.e., Auto-sklearn [7], NASNet [9] and ExploreKit](#bookmark15) [[11]](#bookmark12). We will show in detail how these methods follow the basic framework of AutoML proposed in Section [2.2.2.](#bookmark50)

**7.1 Model Selection using Auto-sklearn**

As each learning problem has its own preference over learn- [ing tools [1], suitable machine learning models should be](#bookmark1) chosen speciﬁcally for each learning problem. Automated model selection is a very typical application of AutoML

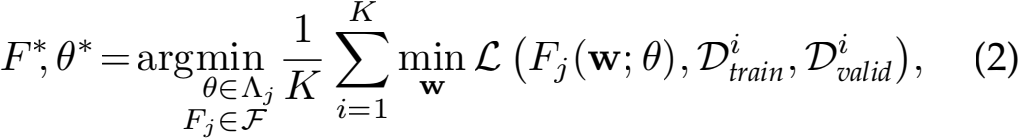
(Section [3.2)](#bookmark94). In the context of supervised classiﬁcation, the purpose is to automatically select best classiﬁers and setup their hyper-parameters properly.

An representative example of automated modelselection [approach is Auto-sklearn [7], which is built on Scikit-Learn](#bookmark8) [[14] package](#bookmark14). Some widely used classiﬁers and their hyper- parameters are listed in Table [4.](#bookmark95) In Auto-sklearn, model selection is formulated as a CASH problem (Example [1),](#bookmark189) which aims to minimize the validation loss with respect to the model as well as its hyper-parameters and parameters.

[**Example 1** (CASH Problem [7], [39])](#bookmark8)**.** Let F = {F1 , · · · , FR } be a set of learning models, and each model has hyper-parameter θj with domain Λj , Dtrain = {(**x**1 , y1 ) , · · · , (**x**n , yn )} be a training set which is split into K cross-validation

folds {Dain , · · · , Dtin } and {Dalid , · · · , Did } with Drain ∪

Dalid = Dtrain for i = 1, . . . , K. Then, the Combined Algorithm Selection and Hyper-parameter (CASH) optimization problem is deﬁned as



where L(Fj (**w**j ; θj ), Drain , Dalid ) denotes the loss that Fj achieves on Dalid with parameter **w**j , hyper-parameter θj and training data Drain.

As discussed in Section [3.2.2, in the model selection](#bookmark96) problem, a candidate conﬁguration comprises a classiﬁer in Scikit-Learn and its hyper-parameters, and the search space is spanned by conﬁgurations of this kind. However, [(2) is very hard to optimize](#bookmark191). First, since there is no explicit expression for the objective, we do not know its properties, e.g., the degree of smoothness, which can be very helpful for traditional optimization problems. In addition, the decision variables,i.e., θ and Fj , may not even be continuous, e.g., θ for kNN includes the number of nearest neighbors that is a discrete variable. Finally, in order to estimate the validation loss, we need to train the model Fj and update its parameter **w**j , which is usually very expensive.

[In [39], sequential model-based algorithm conﬁguration](#bookmark53) [(SMAC) [105], a tree-based Bayesian optimization method,](#bookmark192) [was used as the optimizer to solve (2)](#bookmark191). Then, for the eval- uator, the basic method, i.e., direct evaluation was used. Besides, meta-learning was employed as the experienced technique to get better initialization (i.e., warm-starting conﬁguration generation in Section [6.1.3)](#bookmark178). Finally, rather than discarding models searched in the conﬁguration space, Auto-sklearn stores them and to use a post-processing method to construct an ensemble of them. This automatic

18

TABLE 10

Illustration of how examples Section [7](#bookmark31)fall into the proposed framework in Figure [6.](#bookmark52) 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 [[7]](#bookmark8) | SMAC [[105] algorithm (warm-start](#bookmark192)  by meta-learning) | direct evaluation (train model parameter  with optimization algorithms) | out-of-box classiﬁers |
| NASNet [[9]](#bookmark10) | recurrent neural networks (trained with REINFORCE algorithm [[175])](#bookmark193) | direct evaluation (train child network with stochastic gradient descent) | convolutional neural networks |
| ExploreKit [[11]](#bookmark12) | greedy search algorithm | classiﬁers trained with meta-features | subsequent learning models |

TABLE 11

Performance comparison between Auto-sklearn (ASK) and other 15 classiﬁers (C1-C15) within its conﬁguration space. OpenML data sets (their

IDs are in the ﬁrst column) are used and median balanced error (BER) are reported. BER of Auto-sklearn is obtained based on an ensemble of top

classiﬁers it searched in conﬁguration space; and BERs of C1-C15 are obtained by ﬁne-tuning each classiﬁer separately. Bold number indicates

the best BER as well not those statistical signiﬁcant ones.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | ASK | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | C12 | C13 | C14 | C15 |
| 38 | **2.15** | **2.68** | 50.22 | **2.15** | 18.06 | 11.22 | **1.77** | 50.00 | 8.55 | 16.29 | 17.89 | 46.99 | 50.00 | 8.78 | 2.34 | 15.82 |
| 46 | **3.76** | 4.65 | — | 5.62 | 4.74 | 7.88 | **3.49** | 7.57 | 8.67 | 8.31 | 5.36 | 7.55 | 9.23 | 7.57 | 4.20 | 7.31 |
| 179 | **16.99** | **17.03** | 19.27 | 18.31 | **17.09** | 21.77 | **17.00** | 22.23 | 18.93 | 17.30 | 17.57 | 18.97 | 22.29 | 19.06 | 17.24 | **17.01** |
| 184 | **10.32** | **10.52** | — | 17.46 | **11.10** | 64.74 | **10.42** | 31.10 | 35.44 | 15.76 | 12.52 | 27.13 | 20.01 | 47.18 | **10.98** | 12.76 |
| 554 | **1.55** | 2.42 | — | 12.00 | 2.91 | 10.52 | 3.86 | 2.68 | 3.34 | 2.23 | **1.50** | 10.37 | 100.00 | 2.75 | 3.08 | 2.50 |

ensemble construction makes the ﬁnal result more robust against overﬁtting.

Table [11](#bookmark194) [(from Table 3 in [7]) shows the performance of](#bookmark8) Auto-sklearn (denoted as “ASK”), where classiﬁers C1-15 denotes: Adaboost, Bernoulli naive Bayes, decision tree, ex- treme random trees, Gaussian naive Bayes, gradient boost- ing, KNN, LDA, linear SVM, kernelSVM, multinomial naive Bayes, passive aggressive, QDA, random forest, and linear classiﬁer, respectively. As we can see, Auto-sklearn consis- tently ﬁnds the best conﬁguration in its search space, which demonstrates the success of AutoML in model selection.

**7.2 Reinforcement Learning for NAS (NASNet)**

As mentioned in Section [1,](#bookmark15) since the success of AlexNet on [image classiﬁcation of ImageNet data set [17], the design of](#bookmark16) new neural architectures has become the main means to get better predicting performance in the deep learning domain. This raises the research interests in automatic searching [network architectures for given tasks [9], [15], [16]](#bookmark10).

Taking CNN as an example, the design choices for each convolution layer are listed in Figure [11.](#bookmark105) A conﬁguration (architecture design) contains designs of all convolution layers in a CNN, which leads to a very large search space. The common approach is to build the architecture layer-by- layer. As shown in Figure [20](#bookmark196) one conﬁguration in the NAS problem is a sequence of design decisions. However, in a CNN architecture, the effect of lower layer design depends [on those of higher ones [81], [130]](#bookmark104). This makes the ﬁnal performance of the whole architecture a delayed reward. Motived by such facts, RL is employed in NAS to search for [a optimal sequence of design decisions [16], [64]](#bookmark197). Besides, the direct evaluation is used in these works as the evaluator.

As RL is slow to converge, to make the search faster, transfer learning, which is ﬁrstly used to cut the search [space into several blocks, was developed in [23], [64]; then, a](#bookmark74) [parameter sharing scheming is proposed in [115] to further](#bookmark134) narrow down the search space; and greedy search has also [been considered as a replacement for RL [10], [116]](#bookmark11).

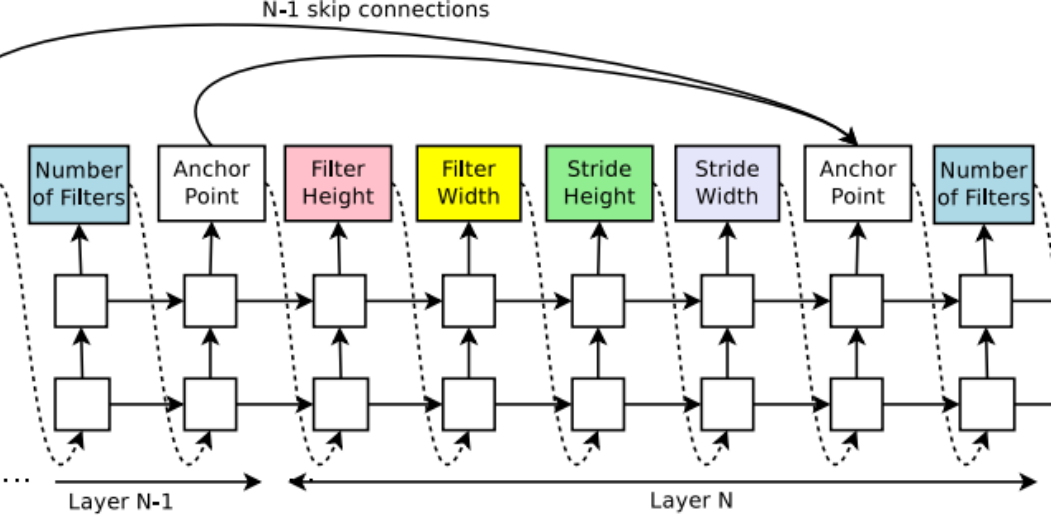


Fig. 20. An illustration of the multi-step decision process of generating a conﬁguration for one convolutional layer using recurrent neural network (RNN) (the image is from [[22]), where anchor point is used to deal with](#bookmark75) skip connections.

Table [12](#bookmark195) presents a comparison between human- designed CNN and those searched by NAS. As can be seen, through AutoML, CNN with less depth and fewer parameters with comparable performance to that of state- of-art CNN designed by humans.

TABLE 12

Comparison of some commonly used human-designed networks with

recently proposed NAS methods on CIFAR-10 data set. Parameters are

in millions and error are measured by top-1 accuracy (in percentage).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | network | depth | parameters | error |
| human- designed | AlexNet [[176]](#bookmark198) | 5 | 0.4M | 11.21% |
| VGG16 [[19]](#bookmark18) | 16 | 138M | 6.44% |
| ResNet [[3]](#bookmark3) | 1001 | 10.2M | 4.62% |
| DenseNet [[21]](#bookmark19) | 190 | 25.6M | 3.46% |
| automated by NAS | NASNet [[9]](#bookmark10) | 39 | 37.4M | 3.65% |
| NASNet-A [[23]](#bookmark74) | — | 27.6M | 2.40% |
| Block-QNN [[64]](#bookmark83) | 22 | 39.8M | 3.54% |
| ENAS [[115]](#bookmark134) | 20 | 4.6M | 2.89% |
| PNAS [[10]](#bookmark11) | 15 | 3.2M | 3.41% |

**7.3 Feature Construction using ExploreKit**

One of the most representative works in automatic feature [construction is ExploreKit [11]](#bookmark12). It aims to generate new

19

features to improve the performance of the learning tools. In this setting, a candidate conﬁguration is a set of generated features. Figure [21](#bookmark200) shows the system architecture of Ex- ploreKit. Its main body of the is an iterative process, where each iteration comprises three steps, candidate feature gen- eration, candidate feature ranking, candidate feature evalu- ation and selection. It is by instinct a greedy search strategy since the search space is extremely large. Additionally, meta- learning techniques are employed in the ranking step to fast estimate the usefulness of candidate features and accelerate the subsequent evaluation step.

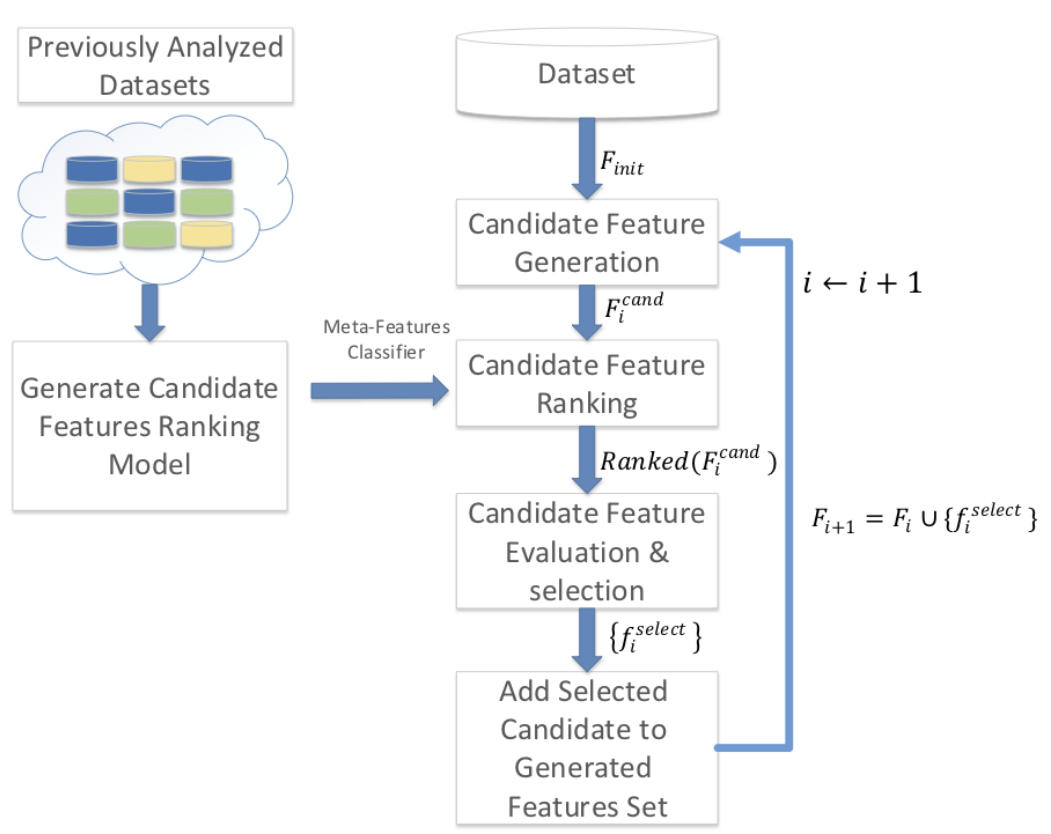


Fig. 21. The system architecture of ExploreKit (the image is from [[11])](#bookmark12).

The optimizer of ExploreKit employs a greed rule-based strategy to explore the search space. 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, nor- malization), 2) binary ones (e.g., +, -, ×, ÷), and 3) higher- order ones (e.g., GroupByThenMax, GroupByThenAvg). A pre- determined 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 in- tractable. To address this issue, ahead of the evaluation and selection step, ExploreKit uses meta-learning roughly rank all candidate features. At this step, a ranking classiﬁer, trained with historical knowledge on feature engineering, is used to fast identify promising candidates. In the evaluation step that follows, features predicted more useful will be considered ﬁrst.

Finally, ExploreKit conducts more expensive and accu- rate evaluations on the candidate features. Error reduction on the validation set is used as the metric for feature importance. Candidate features are evaluated successively 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.

TABLE 13

Experimental results of ExploreKit [[11] on ten representative data sets](#bookmark12).

This table reports the percentage of error reduction brought by the

features generated with ExploreKit. Three base machine learning

models are considered, namely decision tree (DT), support vector

machine (SVM), and random forest (RF).

|  |  |  |  |
| --- | --- | --- | --- |
| data set | DT | SVM | RF |
| CPU act | 23.2% | 23.6% | 13.9% |
| CPU small | 26% | 25.1% | 13.6% |
| Delta elevators | 24.1% | 0.8% | 7.6% |
| Mammography | 47.51% | 45.7% | 40.6% |
| Ailerons | 30.5% | 2.4% | 49.6% |
| Web Data | 1.7% | 10.3% | 0.27% |
| Bank marketing | 29.5% | 10.3% | 31.4% |
| Vehicle IT | 1.6% | 10.6% | 1.3% |
| Vehicle Norm | 11.4% | 10.9% | 9.4% |
| Poker | 57.5% | 8.34% | 13.2% |
| **Average** | **25.3%** | **14.8%** | **18.1%** |
| **(Median)** | **(25.1%)** | **(10.5%)** | **(13.4%)** |

Table [13](#bookmark199) presents the improvement ExploreKit achieves on ten representative data sets. The performance of the generated features depends on the data set and the base classiﬁer, and some encouraging results have been observed.

**8 SUMMARY**

In this section, we ﬁrst review the history of AutoML (Section [8.1), then brieﬂy summarize how its current status](#bookmark201) in the academy and industry (Section [8.2), ﬁnally discuss its](#bookmark202) future works (Section [8.3)](#bookmark54).

**8.1 A Brief History of AutoML**

As mentioned in Section [1, the idea of AutoML emerged](#bookmark15) long before it was formally proposed as a new concept [in Auto-weka [39] and ICML-2014’s AutoML workshop](#bookmark53) [[177]](#bookmark203). Besides, automation of different learning processes in AutoML, i.e., feature engineering, model selection and algorithmselection, as shown in Figure [1, has also been in](#bookmark7)- dividually visited by many researchers in ﬁelds of machine learning, data mining and artiﬁcial intelligence. However, AutoML only becomes practical and a big focus recently, due to the big data, the increasing computation of modern computers, and of course, the great demand of machine learning application.

On the feature level, feature selection is a traditional topic in machine learning, it tries to automatically remove unnecessary features making learning models simpler and [more interpretable [178]](#bookmark204). Many methods have been pro- [posed and Lasso [179] is a landmark of them](#bookmark205). Dimension [reduction methods, such as PCA [68] and LDA [69], have](#bookmark89) also been popularly used to deal with high-dimensionalfea- tures. These methods tries to ﬁnd a better representation for the input data. However, the size of the reduced dimension needs human speciﬁcations. During 1990s, many attempts were done to automatically construct better features based [on genetic algorithms [86], [87], [88]. However, only recently](#bookmark206) did automated learning feature representations become pos- sible for some structured data, such as CNN for images and [RNN for sequential data [67]](#bookmark87).

Model selection is the task of selecting a proper model from a set of candidate models. Theoretical foundation,i.e., [statistical learning theory [13], [180], is ﬁrstly paved for](#bookmark207)

20

TABLE 14

A brief list of AutoML products in the industrials, and “——” indicated no ofﬁcialannouncements are found.

|  |  |  |  |
| --- | --- | --- | --- |
|  | company | AutoML products | customer |
| public company | Google | Deployed in Google’s Cloud | Disney, ZSL, URBN |
| Microsoft | Deployed in Azure |  |
| IBM | IBM Watson Studio |  |
| startup | H2O.ai | H2O AutoML Package | AWS, databricks, IBM, NVIDIA |
| Feature Labs | Feature Labs’ platform | NASA, MONSANTO, MIT, KOHL’S |
| 4Paradigm | AutoML platform | Bank of China, PICC, Zhihu |

model selection, which showshow 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 proper values 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 [181], [182] and gradient-based](#bookmark210) [optimization [121], have been considered for ﬁnding hyper](#bookmark137)- parameters. These methods have strong assumptions on the optimization model, which limits their application in practice. These motivate many tailormade methods for the selection of various machine learning models, such as the [kernel selection for SVM [183], learning K for K-means [184]](#bookmark211) and genetic programming for neural networks architectures [[93], [94]](#bookmark123). Finally, as a single model may not be strong enough, conﬁguring an ensemble of models is considered [in [185]](#bookmark212).

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 [[56], [186]](#bookmark71). 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. In the past, convex and simple models, such as logistic regression and SVM, are widely used in machine learning, and the data sets are also not large. At that time, algorithmselection is not an important problem, and it is easy to ﬁnd good parameters [for a given model [187]](#bookmark213). Recently, as the data gets larger, and complex models, e.g., deep networks, become popular, the optimization algorithms can have an important impact on [the generalization performance [75], [76], [188]](#bookmark99). Beside, more hyper-parameters are also involved in these algorithms, e.g., [Adagrad [189] has more hyper-parameters than plain SGD](#bookmark214) [[75], which make them harder to tune](#bookmark99).

**8.2 Current Status**

Nowadays,in academy, AutoML is a very complex problem and also an extremely active research area, and there are many new opportunities and problems in AutoML that are not visited in the above history, e.g., usage of experienced techniques (Section [6)](#bookmark59). Lots of papers focusing on AutoML appear on various conferences and journals, such as ICML, NIPS, KDD, AAAI, IJCAI and JMLR (see the reference list) and some of their open-source projects are extremely pop- ular on GitHub (Table [15); many workshops are organized,](#bookmark209) [such as AutoML workshop at ICML from 2014 to 2018 [177],](#bookmark203)

[[190], [191], [192], [193]; some competitions such as, AutoML](#bookmark215) [Challenge at PAKDD [194] and NIPS [195], are hold as well](#bookmark216).

TABLE 15

Some popular open-source research projects on Github (up to Nov.

2018). More stars indicates greater popularity.

|  |  |  |  |
| --- | --- | --- | --- |
| Project | stars | Project | stars |
| TPOT | 4326 | hyperopt | 2302 |
| autokeras | 3728 | adanet | 1802 |
| H2O AutoML | 3262 | darts | 1547 |
| Auto-sklearn | 2367 | ENAS-pytorch | 1297 |
| MOE | 1077 | Spearmint | 1124 |

In industry, many products of AutoML are also available. Some examples are listed in Table [14.](#bookmark208) All these compa- nies try to develop an end-to-end AutoML pipeline (Fig- ure [1), but with different focuses](#bookmark7). For example, Feature Labs targets at feature engineering; NAS is built in Google’s Cloud to help design deep networks for computer vision applications. All these products signiﬁcantly reduce their customers’ efforts in deploying machine learning tools for real applications.

**8.3 Future Works**

First of all, as AutoML focuses on how to do machine learning in a special way (Deﬁnition [2), the current trends](#bookmark45) in machine learning can also be seen as future works of AutoML. [Examples are human interpretability [196] and](#bookmark217) [privacy [197] in machine learning](#bookmark218). In the rest of this section, we focus more on future works that are closely related to the framework proposed in Section [2.2.](#bookmark37)

*8.3.1 Problem setup*

How to create features from the data is a fundamental problem not only in machine learning, but also many related areas. For example, scale-invariant feature transform (SIFT) [[198] and histograms of oriented gradients (HoG) [199], suc](#bookmark219)- cessfully generalized for many problems and applications in computer vision. Similarly, in natural language processing (NLP), “term frequency-inverse document frequency” (TF- [IDF) [200], is easy to calculate and performs well across](#bookmark220) many NLP tasks.

As what data should be used for the subsequent learning tools heavily depends on application scenarios (Figure [1),](#bookmark7) 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

21

RNN. More recently, some endeavors have been made for [relational data set, i.e., DSM [12] (Table](#bookmark13) [2)](#bookmark44). Their success lies on utilizing the common relationships inside the data. Besides, with those automatically generated features, the performance of subsequent learning tools are signiﬁcantly improved. Thus, one important future work is to automati- cally creating features from the data.

*8.3.2 Techniques*

**Basic techniques:** In Section [2.2, we have proposed a](#bookmark37) framework, where conﬁgurations are updated based on alternative iterations between the optimizer and evaluator. However, AutoML can be extremely resource-consuming because of the extensive and complex search space, and the expensive evaluation. For example, 800 GPUs and 28 [days are used in [16] for NAS with reinforcement learn](#bookmark197)- ing to discover the convolutional architecture on CIFAR-10 data set. Thus developing more efﬁcient basic techniques is always desired. Higher efﬁciency can be achieved by either proposing algorithms for the optimizer, which visit less conﬁgurations before reaching a good performance, or designing better methods for the evaluator, which can offer more accurate evaluations but in less time.

One interesting direction is to simultaneously optimize conﬁgurations and parameters, such methods have been re- [cently explored in NAS [46], [201] and automated searching](#bookmark145) [step-size for SGD [202], which have shown to be much more](#bookmark221) efﬁcient than previous state-of-the-arts.

**Experienced techniques:** Meta-learning has been widely used to facilitate AutoML. However, there are some con- siderations for using meta-learning, which also indicate the orientation for future study, for example: how to better char- acterize 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 Au- toML, how to automate meta-learning is also an interesting and meaningful research topic.

Transfer learning has found its successful applications mainly in NAS. We are looking forward to more transfer learning techniques employed in a wider scope of AutoML. Also, it has been well realized that knowledge transfer does not always offer improvement. One research topic [of transfer learning is “negative transfer” [31], where the](#bookmark24) 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.

*8.3.3 Applications*

In this survey, we have focused on the supervised-learning problem. It is also the most considered problem in AutoML, due to the learning performance can be clearly evaluated. However, AutoML can be applied in many other problems in machine learning as well. For example, recently, AutoML [approaches have been applied in, e.g., active learning [120],](#bookmark136) [neural network compression [203], and semi-supervised](#bookmark222) [learning (SSL) [204]](#bookmark223).

While the framework in Section [2.2](#bookmark37) and techniques in Section [5-6](#bookmark58)can still be applied in these problems, as different

learning tools have to be used, the search space is different. Besides, properties underneath these problems should be further and carefully explored in order to achieve good performance. [Taking semi-supervised learning [204] as an](#bookmark223) example. Existing AutoML for supervised learning can not well address SSL problems. The reason is that SSL intro- duces new challenges. Speciﬁcally, the feature engineering is much harder and the performance is much more sen- sitive, due to the limited labeled data. Thus, appropriate meta-features (perhaps unsupervised feature is one possible solution) and safeness are crucial for automated SSL to boost the performance upper-bound and lower-boundsimultane- ously.

*8.3.4 Theory*

**Optimization theory:** As shown in Figure [6, the search](#bookmark52)- ing process of AutoML can be considered as a black box optimization problem, where the underneath optimization function is measured by the evaluator. While many theories of convergence have been developed for basic techniques in Section [4, e.g., derivative free optimization [84], [108], gradi](#bookmark30)- [ent descent [121], and greedy search [125], it is still not clear](#bookmark137) how fast they can identify a good conﬁguration. However, convergence speed is a critical problem for AutoML and matters a lot in the choice of techniques for the optimizer, as the evaluation of one conﬁguration usually requires a model training, which is very expensive.

**Learning theory:** On this topic, we ﬁrst care about which type of problems can or cannot be addressed by an AutoML approach. In Section [2.1, we have shown that it is not possi](#bookmark46)- ble for an AutoML approach to achieve good performance on all learning problems due to No Free Lunch theorems [stated in [5] and [6]](#bookmark6). However, AutoML attempts to obtain good performance across learning tasks and datasets. Thus, it is interesting to ﬁgure out which type of learning problems can AutoML deal with and what assumptions can we made on these learning problems.

After determine whether the problem can be addressed by AutoML, for a speciﬁc AutoML approach, it is also important to clarify its generalization ability, i.e., “how much training data is sufﬁcient? what general bounds can be found to relate the conﬁdence of the learned hypotheses to the amount of training experience and characters of learner’s hypothesis space? [[1]”](#bookmark1) . This will help us better design AutoML approaches and understand how these approaches can generalize from seen to unseen data. [AdaNet [205] is a pioneering work](#bookmark224) towards this direction, it analyzed the generalization ability of all possible architectures in a NAS problem and used the derived bound to guide a simple optimization technique to search for architecture.

**9 CONCLUSION**

Motivated by the academic dream and industrial needs, the automated 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 to show how these approaches are realized. We also provide taxonomies of existing works based on “what” and “how”

22

to automate,which acts as a guidance to design new and use old AutoML approaches. We further discuss how existing works can be organized according to our taxonomies in detail. Finally, we brieﬂy review the 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.

**ACKNOWLEDGE**

First, we would like to thank Prof. Isabelle Guyon (UPSud, INRIA, University Paris-Saclay), Prof. Hugo Jair Escalante (INAOE), and Mr. Zhengyin Liu (INRIA) for their valuable comments on our draft version. Furthermore, we would also like to thank, Prof. Bogdan Gabrys (UTS), Dr. Bo Han (UTS, RIKEN), Mr. Saket Maheshwary (IIIT, Hyderabad), Ms. Yaqing Wang (HKUST), and Mr. Yihui He (CMU), for their suggestion on the public version of the survey.

**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. Panneershel- vam, 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] D. H. Wolpert, “The lack of a priori distinctions between learning algorithms,” Neural computation, vol. 8, no. 7,pp. 1341–1390, 1996.

[6] D. H. Wolpert and W. G. Macready, “No free lunch theorems for optimization,” IEEE Transactions on Evolutionary Computation, vol. 1, no. 1,Apr. 1997.

[7] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, “Efﬁcient and robust automated machine learn- ing,” in Advances in Neural Information Processing Systems, 2015, pp. 2962–2970.

[8] L. Kotthoff, C. Thornton, H. Hoos, F. Hutter, and K. Leyton- Brown, “Auto-WEKA 2.0: Automatic modelselection and hyper- parameter optimization in WEKA,” Journal of Machine Learning Research, vol. 18, no. 1, pp. 826–830, 2017.

[9] B. Zoph and Q. V. Le, “Neural architecture search with reinforce- ment learning,” in International Conference on Learning Representa- tions, 2017.

[10] 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.

[11] 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.

[12] J. M. Kanter and K. Veeramachaneni, “Deep feature synthesis: To- wards automating data science endeavors,” in IEEE International Conference on Data Science and Advanced Analytics, 2015, pp. 1–10.

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

[14] 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.

[15] L. Xie and A. Yuille, “Genetic CNN,” in International Conference on Computer Vision. IEEE, 2017, pp. 1388–1397.

[16] B. Baker, O. Gupta, N. Naik, and R. Raskar, “Designing neural network architectures using reinforcement learning,” in Interna- tional Conference on Learning Representations, 2017.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet: classi- ﬁcation with deep convolutional neural networks,” in Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.

[18] 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.

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

[20] 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.

[21] 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.

[22] I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le, “Neural optimizer search with reinforcement learning,” in International Conference on Machine Learning, 2017.

[23] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning trans- ferable architectures for scalable image recognition,” in Conference on Computer Vision and Pattern Recognition, 2017.

[24] S. Huang, X. Li, Z. Cheng, Z. Zhang, and A. G. Hauptmann, “GNAS: A greedy neural architecture search method for multi- attribute learning,” in ACM Multimedia, 2018, pp. 2049–2057.

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

[26] H. Cai,J. Yang, W. Zhang, S. Han, and Y. Yu, “Path-level network transformation for efﬁcient architecture search,” in International Conference on Machine Learning, 2018.

[27] M. J. Smith, R. Wedge, and K. Veeramachaneni, “FeatureHub: To- wards collaborative data science,” in IEEE International Conference on Data Science and Advanced Analytics, 2017, pp. 590–600.

[28] J. Vanschoren, “Understanding machine learning performance with experiment databases,” lirias. kuleuven. be, no. May, 2010.

[29] 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.

[30] J. Vanschoren, “Meta learning,” 2018. [Online]. Avail- able: [https://www.ml4aad.org/wp-content/uploads/2018/09/](https://www.ml4aad.org/wp-content/uploads/2018/09/chapter2-metalearning.pdf) [chapter2-metalearning.pdf](https://www.ml4aad.org/wp-content/uploads/2018/09/chapter2-metalearning.pdf)

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

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

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

[34] M. Elad and M. Aharon, “Image denoising via sparse and redun- dant representations over learned dictionaries,” IEEE Transactions on Image processing, vol. 15, no. 12, pp. 3736–3745, 2006.

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

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

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

[38] C. L. Phillips and R. D. Habor, Feedback control systems. Simon &

Schuster, 1995.

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

[40] 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 Learn- ing, 2013, pp. I–115.

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

[42] E. R. Sparks, A. Talwalkar, D. Haas, M. J. Franklin, M. I. Jordan, and T. Kraska, “Automating model search for large scale machine

23

learning,” in ACM Symposium on Cloud Computing, 2015, pp. 368– 380.

[43] J. N. van Rijn, S. M. Abdulrahman, P. Brazdil, and J. Vanschoren, “Fast algorithmselection using learning curves,” in International symposium on intelligent data analysis. Springer, 2015,pp. 298–309.

[44] Y.-F. Li, H. Wang, T. Wei, and W.-W. Tu, “Derivative-free op- timization via classiﬁcation,” in AAAI Conference on Artiﬁcial Intelligence, vol. 16, 2016, pp. 2286–2292.

[45] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, “SMASH: one- shot model architecture search through hypernetworks,” in Inter- national Conference on Learning Representations, 2018.

[46] H. Liu, K. Simonyan, and Y. Yang, “DARTS: Differentiable archi- tecture search,” arXiv preprint arXiv:1806.09055, 2018.

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

[48] 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.

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

[50] M. G. Smith and L. Bull, “Genetic programming with a genetic algorithm for feature construction and selection,” Genetic Pro- gramming and Evolvable Machines, vol. 6, no. 3, pp. 265–281, 2005.

[51] 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.

[52] F. Nargesian, H. Samulowitz, U. Khurana, E. B. Khalil, and D. Turaga, “Learning feature engineering for classiﬁcation,” in International Joint Conference on ArtiﬁcialIntelligence, vol. 17, 2017, pp. 2529–2535.

[53] 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.

[54] 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.

[55] C. J. Merz, “Dynamical selection of learning algorithms,” in Learning from Data. Springer, 1996, pp. 281–290.

[56] S. Kadioglu, Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann, “Algorithm selection and scheduling,” in Interna- tional Conference on Principles and Practice of Constraint Program- ming, 2011, pp. 454–469.

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

[58] B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Frchette,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.

[59] T. Domhan,J. T. Springenberg, and F. Hutter, “Speeding up auto- matic hyperparameter optimization of deep neural networks by extrapolation of learning curves,” in International Joint Conference on Artiﬁcial Intelligence, 2015.

[60] D. Ha, A. Dai, and Q. V. Le, “Hypernetworks,” in International Conference on Learning Representations, 2016.

[61] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hut- ter, “Towards automatically-tuned neural networks,” in Workshop on Automatic Machine Learning, 2016, pp. 58–65.

[62] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A.

Rusu, A. Pritzel, and D. Wierstra, “Pathnet: Evolution chan- nels gradient descent in super neural networks,” arXiv preprint arXiv:1701.08734, 2017.

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

[64] Z. Zhong, J. Yan, and C.-L. Liu, “Practical block-wise neural network architecture generation,” in IEEE Conference on Computer Vision and Pattern Recognition, 2017.

[65] B. Deng, J. Yan, and D. Lin, “Peephole: Predicting network per- formance before training,” arXiv preprint arXiv:1712.03351, 2017.

[66] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, “Efﬁcient architec- ture search by network transformation,” in AAAI Conference on Artiﬁcial Intelligence, 2018.

[67] 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.

[68] 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.

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

[70] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Ex- tracting and composing robust features with denoising autoen- coders,” in International Conference on Machine learning, 2008, pp. 1096–1103.

[71] H. T. Lam,J.-M. Thiebaut,M. Sinn, B. Chen, T. Mai, andO. Alkan, “One button machine for automating feature engineering in relational databases,” arXiv preprint arXiv:1706.00327, 2017.

[72] A. Kaul, S. Maheshwary, and V. Pudi, “Autolearnautomated feature generation and selection,” in IEEE International Conference on Data Mining, 2017, pp. 217–226.

[73] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, “Decon- volutional networks,” in IEEE Conference on Computer Vision and Pattern Recognition, 2010, pp. 2528–2535.

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

[75] L. Bottou and O. Bousquet, “The tradeoffs of large scale learn- ing,” in Advances in Neural Information Processing Systems, 2008, pp. 161–168.

[76] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, “On optimization methods for deep learning,” in Interna- tional Conference on International Conference on Machine Learning, 2011, pp. 265–272.

[77] D. C. Liu and J. Nocedal, “On the limited memory BFGS method for large scale optimization,” Mathematical Programming, vol. 45, no. 1-3, pp. 503–528, 1989.

[78] M. M. Salvador, M. Budka, and B. Gabrys, “Automatic compo- sition and optimization of multicomponent predictive systems with an extended auto-weka,” IEEE Transactions on Automation Science and Engineering, pp. 1–14, 2018.

[79] R. Negrinho and G. Gordon, “Deeparchitect: Automatically de- signing and training deep architectures,” 2018.

[80] 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.

[81] 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.

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

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

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

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

[86] 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.

[87] W. Tackett, “Genetic programming for feature discovery and image discrimination,” in International Conference on Genetic Algorithms, 1993.

[88] 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.

[89] R. S. Olson, N. Bartley, R. J. Urbanowicz, and J. H. Moore, “Eval- uation of a tree-based pipeline optimization tool for automating data science,” in Genetic and Evolutionary Computation Conference. ACM, 2016, pp. 485–492.

[90] R. S. Olson and J. H. Moore, “TPOT: A tree-based pipeline optimization tool for automating machine learning,” in Workshop on Automatic Machine Learning, 2016, pp. 66–74.

[91] S.-W. Lin, K.-C. Ying, S.-C. Chen, and Z.-J. Lee, “Particle swarm optimization for parameter determination and feature selection of support vector machines,” Expert Systems with Applications, vol. 35, no. 4, pp. 1817–1824, 2008.

[92] P. R. Lorenzo, J. Nalepa, M. Kawulok, L. S. Ramos, and J. R. Pas- tor, “Particle swarm optimization for hyper-parameter selection

24

in deep neural networks,” in Genetic and Evolutionary Computation Conference. ACM, 2017, pp. 481–488.

[93] X. Yao, “Evolving artiﬁcial neural networks,” Proceedings of the IEEE, vol. 87, no. 9, pp. 1423–1447, 1999.

[94] C. Zhang, H. Shao, and Y. Li, “Particle swarm optimisation for evolving artiﬁcial neural network,” in IEEE International Confer- ence on Systems, Man, and Cybernetics, vol. 4, 2000, pp. 2487–2490.

[95] K. O. Stanley and R. Miikkulainen, “Evolving neural net- works through augmenting topologies,” Evolutionary Computa- tion, vol. 10, no. 2, pp. 99–127, 2002.

[96] 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.

[97] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, “Regularized evolution for imageclassiﬁer architecture search,” arXiv preprint arXiv:1802.01548, 2018.

[98] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, “Hierarchical representations for efﬁcient archi- tecture search,” in International Conference on Learning Representa- tions, 2018.

[99] K. Swersky, J. Snoek, and R. P. Adams, “Freeze-thaw bayesian optimization,” arXiv preprint arXiv:1406.3896, 2014.

[100] T. Nickson, M. A. Osborne, S. Reece, and S. J. Roberts, “Auto- mated machine learning on big data using stochastic algorithm tuning,” arXiv preprint arXiv:1407.7969, 2014.

[101] 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.

[102] J. Snoek, H. Larochelle, and R. P. Adams, “Practical bayesian optimization of machine learning algorithms,” in Advances in neural information processing systems, 2012, pp. 2951–2959.

[103] M. Wistuba, N. Schilling, and L. Schmidt-Thieme, “Scalable gaussian process-based transfer surrogates for hyperparameter optimization,” Machine Learning, vol. 107, no. 1, pp. 43–78, 2018.

[104] J. S. Bergstra, R. Bardenet,Y. Bengio, and B. Kgl, “Algorithms for hyper-parameter optimization,” in Advances in Neural Information Processing Systems, 2011, pp. 2546–2554.

[105] F. Hutter, H. H. Hoos, and K. Leyton-Brown, “Sequential model- based optimization for general algorithm conﬁguration,” in Inter- national Conference on Learning and Intelligent Optimization, 2011, pp. 507–523.

[106] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams, “Scalable bayesian opti- mization using deep neural networks,” in International Conference on Machine Learning, 2015, pp. 2171–2180.

[107] 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.

[108] T. B. Hashimoto, S. Yadlowsky, and J. C. Duchi, “Derivative free optimization via repeated classiﬁcation,” International Conference on ArtiﬁcialIntelligence and Statistics, 2018.

[109] 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.

[110] M. Valko, A. Carpentier, and R. Munos, “Stochastic simultaneous optimistic optimization,” in International Conference on Machine Learning, 2013, pp. 19–27.

[111] C. Andrieu, N. De Freitas, and A. Doucet, “Sequential mcmc for bayesian modelselection,” in IEEE Signal Processing Workshop on Higher-Order Statistics, 1999, pp. 130–134.

[112] K. Swersky, J. Snoek, and R. P. Adams, “Multi-task bayesian op- timization,” in Advances in Neural Information Processing Systems, 2013, pp. 2004–2012.

[113] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. Cox, “Hy- peropt: a python library for modelselection and hyperparameter optimization,” Computational Science & Discovery, vol. 8, no. 1, p. 014008, 2015.

[114] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduc- tion. MIT press, 1998.

[115] 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.

[116] J.-M. Prez-Ra, M. Baccouche, and S. Pateux, “Efﬁcient pro- gressive neural architecture search,” in British Machine Vision Conference, 2018.

[117] A. Gyrgy and L. Kocsis, “Efﬁcient multi-start strategies for local search algorithms,” Journal of Artiﬁcial Intelligence Research, vol. 41, pp. 407–444, 2011.

[118] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Tal- walkar, “Hyperband: A novel bandit-based approach to hyper- parameter optimization,” Journal of Machine Learning Research, vol. 18, no. 1, pp. 6765–6816, 2017.

[119] R. Gaudel and M. Sebag, “Feature selection as a one-player game,” in International Conference on Machine Learning, 2010, pp. 359–366.

[120] 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.

[121] Y. Bengio, “Gradient-based optimization of hyperparameters,” Neural Computation, vol. 12, no. 8, pp. 1889–1900, 2000.

[122] F. Pedregosa, “Hyperparameter optimization with approximate gradient,” in International Conference on Machine Learning, 2016, pp. 737–746.

[123] 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.

[124] J. A. Tropp, “Greed is good: Algorithmic results for sparse approximation,” IEEE Transactions on Information theory, vol. 50, no. 10, pp. 2231–2242, 2004.

[125] F. Bach et al., “Learning with submodular functions: A convex optimization perspective,” Foundations and TrendsoR in Machine Learning, vol. 6, no. 2-3, pp. 145–373, 2013.

[126] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, “Neural architec- ture optimization,” in Advances in Neural Information Processing Systems, 2018, pp. 2546–2554.

[127] Y.-Q. Hu, Y. Yu, W.-W. Tu, Q. Yang, Y. Chen, and W. Dai, “Multi- ﬁdelity automatic hyper-parameter tuning via transfer series expansion,” in AAAI Conference on Artiﬁcial Intelligence, vol. 16, 2019, pp. 2286–2292.

[128] O. Maron and A. W. Moore, “Hoeffding races: Accelerating modelselection search for classiﬁcation and function approxima- tion,” in Advances in Neural Information Processing Systems, 1994, pp. 59–66.

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

[130] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, “Greedy layer-wise training of deep networks,” in Advances in Neural Information Processing Systems, 2007, pp. 153–160.

[131] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the im- portance of initialization and momentum in deep learning,” in International Conference on Machine Learning, 2013, pp. 1139–1147.

[132] K. Eggensperger, F. Hutter, H. H. Hoos, and K. Leyton-Brown, “Efﬁcient benchmarking of hyperparameter optimizers via sur- rogates,” in AAAI Conference on Artiﬁcial Intelligence, 2015.

[133] B. Baker, O. Gupta, R. Raskar, and N. Naik, “Accelerating neural architecture search using performance prediction,” in Interna- tional Conference on Learning Representations, 2018.

[134] J. Vanschoren, “Meta-learning: A survey,” arXiv preprint arXiv:1810.03548, 2018.

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

[136] R. Leite, P. Brazdil, and J. Vanschoren, “Selecting classiﬁcation algorithms with active testing,” in International Workshop on Ma- chine Learning and Data Mining in Pattern Recognition, 2012, pp. 117–131.

[137] 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.

[138] S. Ali and K. A. Smith-Miles, “A meta-learning approach to automatic kernel selection for support vector machines,” Neu- rocomputing, vol. 70, no. 1-3, pp. 173–186, 2006.

[139] 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.

[140] 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.

[141] T. A. Gomes, R. B. Prudncio, C. Soares, A. L. Rossi, and

A. Carvalho, “Combining meta-learning and search techniques to

25

select parameters for support vector machines,” Neurocomputing, vol. 75, no. 1, pp. 3–13, 2012.

[142] M. Reif, F. Shafait, and A. Dengel, “Meta-learning for evolu- tionary parameter optimization of classiﬁers,” Machine Learning, vol. 87, no. 3, pp. 357–380, 2012.

[143] M. Feurer, J. T. Springenberg, and F. Hutter, “Initializing bayesian hyperparameter optimization via meta-learning,” in AAAI Con- ference on Artiﬁcial Intelligence, 2015, pp. 1128–1135.

[144] H. Hoos and K. Leyton-Brown, “An efﬁcient approach for as- sessing hyperparameter importance,” in International Conference on Machine Learning, 2014, pp. 754–762.

[145] M. Wistuba, N. Schilling, and L. Schmidt-Thieme, “Hyperpa- rameter search space pruning–a new component for sequen- tial model-based hyperparameter optimization,” in Joint Euro- pean Conference on Machine Learning and Knowledge Discovery in Databases. Springer, 2015, pp. 104–119.

[146] 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.

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

[148] D. Kifer, S. Ben-David, and J. Gehrke, “Detecting change in data streams,” in International Conference on Very Large Data Bases. VLDB Endowment, 2004, pp. 180–191.

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

[150] A. L. D. Rossi, A. C. Carvalho, and C. Soares, “Meta-learning for periodic algorithmselection in time-changing data,” in Brazilian Symposium on Neural Networks, 2012, pp. 7–12.

[151] A. L. D. Rossi, A. C. P. de Leon Ferreira, C. Soares, and B. F.

de Souza, “MetaStream: A meta-learning based method for peri- odic algorithmselection in time-changing data,” Neurocomputing, vol. 127, pp. 52–64, 2014.

[152] C. Giraud-Carrier, “Metalearning-a tutorial,” 2008.

[153] P. Brazdil, J. Gama, and B. Henery, “Characterizing the applica- bility of classiﬁcation algorithms using meta-level learning,” in European Conference on Machine Learning. Springer, 1994, pp. 83– 102.

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

[155] P. B. de Miranda, R. B. Prudncio, 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.

[156] 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.

[157] C. Taylor, D. Michie, and D. Spiegalhalter, “Machine learning, neural and statistical classiﬁers,” Journal of the American Statistical Association, 1994.

[158] C. Soares and P. B. Brazdil, “Zoomed ranking: Selection of classi- ﬁcation algorithms based on relevant performance information,” in European Conference on Principles of Data Mining and Knowledge Discovery. Springer, 2000, pp. 126–135.

[159] 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.

[160] 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.

[161] 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.

[162] 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.

[163] M. Reif, F. Shafait, and A. Dengel, “Prediction of classiﬁer train- ing time including parameter optimization,” in Annual Conference on ArtiﬁcialIntelligence. Springer, 2011, pp. 260–271.

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

[165] N. Jankowski, “Complexity measures for meta-learning and their optimality,” in Algorithmic Probability and Friends. Bayesian Predic- tion and ArtiﬁcialIntelligence. Springer, 2013, pp. 198–210.

[166] M. Wistuba, N. Schilling, and L. Schmidt-Thieme, “Learning hy- perparameter optimization initializations,” in IEEE International Conference on Data Science and Advanced Analytics, 2015, pp. 1–10.

[167] M. Lindauer and F. Hutter, “Warmstarting of model-based algo- rithmconﬁguration,” in AAAI Conference on Artiﬁcial Intelligence, 2018, pp. 1355–1362.

[168] F. Hutter, H. Hoos, and K. Leyton-Brown, “An efﬁcient approach for assessing hyperparameter importance,” in International Con- ference on Machine Learning, 2014, pp. 754–762.

[169] R. Bardenet, M. Brendel, B. Kgl, and M. Sebag, “Collaborative hyperparameter tuning,” in International Conference on Machine Learning, 2013, pp. 199–207.

[170] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, “Google vizier: A service for black-box optimization,” in ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2017, pp. 1487–1495.

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

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

[173] M. Wistuba, N. Schilling, and L. Schmidt-Thieme, “Hyperparam- eter optimization machines,” in IEEE International Conference on Data Science and Advanced Analytics. IEEE, 2016, pp. 41–50.

[174] C. Wong, N. Houlsby, Y. Lu, and A. Gesmundo, “Transfer learning with neural automl,” in Advances in Neural Information Processing Systems, 2018, pp. 8365–8374.

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

[176] J. Schmidhuber, “Multi-column deep neural networks for image classiﬁcation,” in IEEE Conference on Computer Vision and Pattern Recognition. IEEE Computer Society, 2012, pp. 3642–3649.

[177] “ICML-2014 AutoML workshop,” 2014. [Online]. Available: <https://sites.google.com/site/automlwsicml14/>

[178] 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.

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

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

[181] 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.

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

[183] N.-E. Ayat, M. Cheriet, and C. Y. Suen, “Automatic model se- lection for the optimization of svm kernels,” Pattern Recognition, vol. 38, no. 10, pp. 1733–1745, 2005.

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

[185] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, “Ensem- ble selection from libraries of models,” in International Conference on Machine Learning, 2004, p. 18.

[186] J. Rice, “The algorithmselection problem,” Advances in Computers, vol. 15, p. 65, 1976.

[187] S. Sra, S. Nowozin, and S. Wright, Optimization for machine learn- ing. MIT Press, 2012.

[188] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term de- pendencies with gradient descent is difﬁcult,” IEEE Transactions on Neural Networks, vol. 5, no. 2, pp. 157–166, 1994.

[189] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient meth- ods for online learning and stochastic optimization,” Journal of Machine Learning Research, vol. 12, no. Jul, pp. 2121–2159, 2011.

[190] “ICML-2015 AutoML workshop,” 2015. [Online]. Available: <https://sites.google.com/site/automlwsicml15/>

[191] “ICML-2016 AutoML workshop,” 2016. [Online]. Available: <https://sites.google.com/site/automl2016/>

[192] “ICML-2017 AutoML workshop,” 2017. [Online]. Available: <https://sites.google.com/site/automl2017icml/>

26

[193] “ICML-2018 AutoML workshop,” 2018. [Online]. Available: <https://www.4paradigm.com/competition/pakdd2018>

[194] “PAKDD 2018 data competition: Automatic machine learning challenge 2018,” 2018. [Online]. Available: [https://sites.google.](https://sites.google.com/site/automl2018icml/) [com/site/automl2018icml/](https://sites.google.com/site/automl2018icml/)

[195] “NIPS 2018 challenge: AutoML for lifelong machine learning,” 2018. [Online]. Available: [https://sites.google.com/](https://sites.google.com/site/automl2018icml/) [site/automl2018icml/](https://sites.google.com/site/automl2018icml/)

[196] “ICML-2018 workshop on human interpretability in machine learning,” 2018. [Online]. Available: [https://sites.google.com/](https://sites.google.com/view/whi2018/home) [view/whi2018/home](https://sites.google.com/view/whi2018/home)

[197] “ICML-2018 workshop on privacy in machine learning and artiﬁcial intelligence,” 2018. [Online]. Available: [https:](https://pimlai.github.io/pimlai18/) [//pimlai.github.io/pimlai18/](https://pimlai.github.io/pimlai18/)

[198] D. G. Lowe, “Object recognition from local scale-invariant fea- tures,” in IEEE International Conference on Computer Vision, vol. 2, 1999, pp. 1150–1157.

[199] 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.

[200] 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.

[201] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki,J. Don- ahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, and K. Si- monyan, “Population based training of neural networks,” arXiv preprint arXiv:1711.09846, 2017.

[202] 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.

[203] Y. He,J. Lin,Z. Liu,H. Wang,L.-J. Li, and S. Han, “AMC: AutoML for model compression and acceleration on mobile devices,” in European Conference on Computer Vision, 2018, pp. 784–800.

[204] Y.-F. Li, H. Wang, T. Wei, and W.-W. Tu, “Towards automated semi-supervised learning,” in AAAI Conference on ArtiﬁcialIntel- ligence, 2019.

[205] C. Cortes, X. Gonzalvo, V. Kuznetsov, M. Mohri, and S. Yang, “AdaNet: Adaptive structural learning of artiﬁcial neural net- works,” in International Conference on Machine Learning, 2017, pp. 874–883.