Artificial Intelligence In Medicine 104 (2020) 101822

Contents lists available at ScienceDirect

Artificial Intelligence In Medicine

journal homepage: www.elsevier.com/locate/artmed

Automated machine learning: Review of the state-of-the-art and opportunities for healthcare

Jonathan Waring[a,b,*], Charlotta Lindvall[c,d], Renato Umeton[a,b,e]

[a] Dana-Farber Cancer Institute, Department of Informatics & Analytics, Boston, MA, 02215, United States
[b] Harvard T.H. Chan School of Public Health, Department of Biostatistics, Boston, MA, 02215, United States
[c] Dana-Farber Cancer Institute, Department of Pyschosocial Oncology and Palliative Care, Boston, MA, 02215, United States
[d] Brigham and Women's Hospital, Department of Medicine, Boston, MA, 02215, United States
[e] Massachusetts Institute of Technology, Cambridge, MA, 02139, United States

ARTICLE INFO

Keywords:
Machine learning
Deep learning
Automated machine learning
AutoML
Healthcare

ABSTRACT

Objective: This work aims to provide a review of the existing literature in the field of automated machine learning (AutoML) to help healthcare professionals better utilize machine learning models "off-the-shelf" with limited data science expertise. We also identify the potential opportunities and barriers to using AutoML in healthcare, as well as existing applications of AutoML in healthcare.

Methods: Published papers, accompanied with code, describing work in the field of AutoML from both a computer science perspective or a biomedical informatics perspective were reviewed. We also provide a short summary of a series of AutoML challenges hosted by ChaLearn.

Results: A review of 101 papers in the field of AutoML revealed that these automated techniques can match or improve upon expert human performance in certain machine learning tasks, often in a shorter amount of time. The main limitation of AutoML at this point is the ability to get these systems to work efficiently on a large scale, i.e. beyond small- and medium-size retrospective datasets.

Discussion: The utilization of machine learning techniques has the demonstrated potential to improve health outcomes, cut healthcare costs, and advance clinical research. However, most hospitals are not currently deploying machine learning solutions. One reason for this is that health care professionals often lack the machine learning expertise that is necessary to build a successful model, deploy it in production, and integrate it with the clinical workflow. In order to make machine learning techniques easier to apply and to reduce the demand for human experts, automated machine learning (AutoML) has emerged as a growing field that seeks to automatically select, compose, and parametrize machine learning models, so as to achieve optimal performance on a given task and/or dataset.

Conclusion: While there have already been some use cases of AutoML in the healthcare field, more work needs to be done in order for there to be widespread adoption of AutoML in healthcare.

1. Introduction

The extensive collection of health data through electronic health records (EHRs), genomic sequencing, and digital health wearables has led to an exponentially growing amount of biomedical "big data" [1–3]. The amount of digital information available to clinicians is becoming simply too much to process: within the timespan of 20 − 40 min that are generally assigned per visit, it is virtually impossible to review 80 + megabytes (equivalent to 20,000 + pages of free text) worth of patient data captured in the average individual EHR [4]. Machine learning, and more recently deep learning, are key techniques that have demonstrated the ability to translate these large health datasets into actionable knowledge. In general, the use of machine learning models could improve patient safety [5–7], improve quality of care [8–10], and reduce healthcare costs [11–13]. Specifically, machine learning has the capability to augment the work of clinicians by processing the billions of patient data points that are stored in EHRs, and it has been successfully applied in many clinical applications already, such as identifying patients at high risk of being transferred to the ICU [14], diagnosing respiratory conditions from chest X-rays [15], detecting early signals of

* Corresponding author at: Dana-Farber Cancer Institute, Department of Informatics & Analytics, Boston, MA, 02215, United States.
E-mail addresses: jonathan_waring@dfci.harvard.edu (J. Waring), charlotta_lindvall@dfci.harvard.edu (C. Lindvall), renato_umeton@dfci.harvard.edu (R. Umeton).

https://doi.org/10.1016/j.artmed.2020.101822
Received 8 October 2019; Received in revised form 17 January 2020; Accepted 17 February 2020

lung cancer [16], and detecting fraudulent and abusive health insurance claims [17]. While machine learning in healthcare is a very active research topic [18,19], most of the health data collected is never used for building predictive models that are successively integrated in the clinical setting [20] with only 15 % of hospitals currently and routinely using machine learning for even limited purposes [21].

While machine learning has a lot of demonstrated benefit, the successful utilization of machine learning requires a large effort from human experts given that no algorithm can achieve good performance on all possible problems (i.e., No Free Lunch [22]). Even though healthcare researchers are familiar with clinical data, they still often lack the machine learning expertise necessary to apply these techniques to big data sources. Healthcare researchers can and do work with expert data scientists [18,19], but the interactive process generally takes a lot of time and effort from both parties. Not only that, but data and human expertise are generally not readily available, especially in healthcare settings [23]. Therefore, it is difficult to devise and deploy machine learning solutions as the whole exercise begins with a lengthy data provisioning process, continues with finding the right collaborators, and involves a continuous back-and-forth between ML experts and domain experts. Automating some of the components requiring human expertise would allow the healthcare industry to more rapidly build, validate, and deploy machine learning solutions, and therefore more readily reap the benefits of improving the quality of health care for patients. Motivated by this goal across industries, AutoML has emerged as a new research field with the goal of automatically optimizing parts of the machine learning pipeline, as shown in Fig. 1.

Different AutoML solutions have emerged in recent years to optimize one or more of these components, several of which are the product of AutoML Challenge competitions between 2015 and 2018. The ChaLearn AutoML Challenges[1] focus on solving supervised machine learning problems without any human intervention given some computational constraints. These computational constraints were slightly different across the challenges, but usually included a time limit (~20 min for training and testing) and memory usage limitations (24GB RAM for the first three rounds, and 56GB thereafter). Some of the competitions included a GPU track, but submissions to these tracks were sparse. The goal of this series of challenges is to create a black box that removes most of the requirements for human expertise in applying machine learning to a wide array of problems, and to help alleviate the potential shortage of data scientists and empower those with domain knowledge. There have already been three series of challenges, each with slightly different problem formulations and datasets, and there is currently an ongoing challenge for temporal relational data,[2] as well as a future challenge for computer vision. A detailed analysis of the AutoML challenges from 2015 to 2018 is reviewed in [24].

Although a formal definition and review of AutoML exists [25], it is aimed at an audience of generalists. Here we will address how AutoML is specifically useful for the healthcare field. We decided to organize this review based on what authors are attempting to automate: automated feature engineering, hyperparameter optimization, pipeline optimizers (addresses more than one component), and neural architecture search. We will look at each of these four categories separately and discuss how they can be applied in a healthcare setting.

2. Methods

We conducted a search for papers published between 2012 and 2019 discussing the field of automated machine learning (AutoML) in four academic journal databases, including Scopus, Google Scholar, Microsoft Academia, and CrossRef using a set of keywords, in disjunction. Specifically, we searched the following keyphrases: "automated machine learning", "automl", "automatic machine learning", "auto machine learning" (see Suppl. Tab. 1 for the resulting AutoML paper collection). We chose to exclude closed-source systems from this review. We also chose to only review papers for which there was available source code or links to project repositories. Papers with limited citation counts (< 5 citations) or AutoML systems with fewer than 10 followers on GitHub were also excluded. One author (JW) reviewed titles and abstracts identified from the database search to verify that a paper actually discussed a topic relevant to the field of AutoML. Identified papers were then read in full and the reference lists were searched for additional sources of review. Additionally, papers were further organized into two topics: classical machine learning algorithms and deep neural networks. There has been substantial work done in both fields, and we have organized our review with dedicated sections describing both subfields of interest. For a detailed count of the papers included and excluded at each stage, refer to Fig. 2.

3. Automated feature engineering

When given a supervised machine learning problem, a data scientist is often tasked with creating explanatory variables, otherwise known as features, that are predictive of the outcome of interest. Successful feature engineering requires the creation of features that not only provide useful insights into the data itself, but also takes into account any limitations of the learning algorithm that is being used. It is important to note that this is not a trivial task, as the performance of a given machine learning algorithm is heavily dependent upon the quality of the input features [26]. The creation of these features often requires extensive domain knowledge, and therefore is usually performed manually by a human expert in a trial-and-error fashion. This makes feature engineering a critical and time-consuming step in the machine learning pipeline.

In order to help alleviate the difficulties that come with feature engineering, automated feature engineering frameworks have emerged with the goal of constructing novel feature sets that improve the performance of subsequent machine learning tools. Notably, platforms like Kaggle[3], Grand Challenges in Biomedical Image Analysis[4], and others, can help alleviete this part too, by outsourcing the required effort via open competitions, but do require lenghtly efforts around data anonymization. Automated feature engineering differs from the field of representation learning [27], which employs deep learning techniques to find a useful feature space for unstructured data types, such as images, text, and audio/video. While representation learning is not considered an AutoML technique, it still plays an important role in the machine learning pipeline for these unstructured data formats, especially in healthcare where it has been shown to provide useful representations of EHR data that facilitates clinical predictive modeling [28,29].

The task of automated feature engineering can be more formally explained as follows. Given a set of m features, $F = [f_1, f_2, ..., f_m]$, a target vector, y, and a machine learning algorithm, $\mathscr{M}$, let $P_{\mathscr{M}}(F, y)$ reflect the model performance on the given features and target vector. Let us also consider k transformation functions, $t_1, t_2, ..., t_k$, and a sequence of transformations $s = t_1(t_2...(f_i))$. Our goal is to find a set of sequences of transformations $S = [s_1, ..., s_r]$ to produce $F_{new} = F' + S$ where $F' \subset F$ which satisfies

$$\underset{S}{\operatorname{argmax}}\, P_{\mathscr{M}}(F_{new}, y)$$

Each set of transformations requires training and evaluating a model in order to verify its performance, and therefore it is clearly computationally infeasible to verify all possibilities. Several different approaches have emerged over the last couple of years in order to deal

[1] http://automl.chalearn.org/
[2] https://www.4paradigm.com/competition/kddcup2019
[3] https://www.kaggle.com/
[4] https://grand-challenge.org/

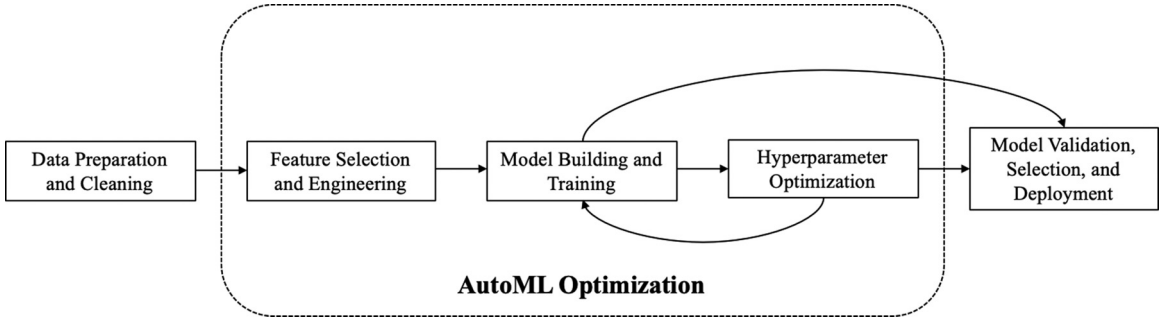

Fig. 1. Typical components of a machine learning problem pipeline. The first step consists of preparing the data. This involves loading and cleaning the data for use in the system, as well as applying any transformations, normalizations, or encodings. The next step involves selecting features to be used in creating the model. This might also involve feature engineering, which is the process of using domain knowledge to create new features to help improve the machine learning model. The next stages involve an iterative process in which one builds, trains, optimizes, validates, and selects a given machine learning algorithm to use for a given problem.

with this issue.

The first approach is the so-called "expand-reduce" method, which was first introduced by Kanter et al. in their development of the Data Science Machine (DSM) [30]. Speaking generally, this method works by applying all feature transformations at once to obtain $[f_1, f_2, ..., f_m] \times [t_1, t_2, ..., t_k] = m \times k$ features. This is then followed by a feature selection step and hyperparameter tuning. The upside of this approach is that there is only one modeling step, excluding the feature selection. However, this approach does not consider compositions of functions and has a performance bottleneck in the feature selection step, given the large number of features to consider. After it was originally introduced, several variations of the "expand-reduce" method have been published, including ExploreKit [31], the One Button Machine (OBM) [32], and most recently AutoLearn [33]. Additionally, the original creators of the DSM have since released Featuretools [34], an open-source implementation of their automated feature engineering algorithm, and FeatureHub [35], a collaborative data science platform in which skilled data scientists may contribute code to perform feature engineering and present experimental results on how crowd-generated features perform with an AutoML model. ExploreKit and AutoLearn also have open source implementations. These algorithms vary slightly in how they perform the feature selection step and in which feature transformations they apply, but there is no clear winner for which one universally performs the best.

A second approach to this problem is to use genetic programming, an evolutionary algorithmic technique. The main idea behind genetic programming is to encode a computer program as an artificial "chromosome" and to evaluate the fitness of this encoding with respect to some pre-defined task, and ideally improve performance over time. In terms of a feature engineering task, Tran et al. [36] proposed genetic programming using a tree-based representation that can be used for both feature construction and implicit feature selection. While this method did provide favorable results in certain experiments, it also provides an unstable solution in which overfitting frequently occurs. However, their method did provide a slight improvement in speed compared to the expand-reduce approach.

Other approaches to the feature engineering problem include a hierarchical organization of transformations, meta learning, and reinforcement learning. Khurana et al. proposed the "Cognito" system [37] which explores various feature construction choices in a hierarchical manner, while progressively maximizing the accuracy of the model through a greedy search strategy. This is done by constructing a directed acyclic transformation graph and applying transformations to all valid input features. This emulates a human trial-and-error process, allowing one to use data-level transformations as logical blocks for measuring performance over time. It has the added benefit of allowing compositions of transformations. The same authors later proposed a similar strategy that employs reinforcement learning [38], whereby a transformation graph is explored via a reinforcement learning agent. The most recent contribution of these authors to the field of automated feature engineering is their "Learning Feature Engineering" (LFE) technique [39] which uses meta learning. LFE works by learning how effective a given transformation (e.g., arithmetic or aggregate operators) on numerical features truly is by learning from past feature engineering experiences. Given a new dataset, LFE recommends a set of useful transformations to be applied without model evaluation or an explicit feature expansion/selection step. This approach uses a substantially lower amount of computational resources and was shown to improve upon previous feature engineering approaches on a majority of the datasets it was tested on. (Table 1)

4. Hyperparameter optimization

Every machine learning model has two types of parameters: hyperparameters that the model designer must manually set prior to training, and normal parameters that are optimized in the training of the model. These hyperparameters are settings that control the behavior of the machine learning algorithm in some way, often in a way that is highly specific to that algorithm. The most basic task of AutoML is to automatically set these hyperparameters to optimize model performance. The performance of most machine learning methods can depend critically upon these hyperparameter settings, and thus it is one of the most important tasks in machine learning [40].

Hyperparameter optimization is often considered an "art", requiring practitioner's experience, general rules of thumb, and sometimes just a brute-force search. In order to make machine learning more accessible to non-technical professionals, computer science researchers have proposed several different automatic hyperparameter selection methods. These methods attempt to quickly find an optimal, or at least an effective, combination of hyperparameter values that maximizes some performance metric for the given machine learning task. These methods are also often given a specified computational resource limit, such as a limited search time or limited memory usage. Using an automatic hyperparameter selection method can greatly reduce the burden on those building a machine learning solution, and several selection algorithms have been shown to find hyperparameter values that are equally good or better than manual tuning by machine learning experts [41,42].

The problem of hyperparameter optimization can be more formally defined as follows. Let $\mathscr{M}$ denote some machine learning algorithm with N hyperparameters. The domain of the i-th hyperparameter is denoted as Λ_i with the overall hyperparameter configuration space being $\Lambda = \Lambda_1, ..., \Lambda_n$. Let any given combination of hyperparameters be $\lambda \in \Lambda$ and the machine learning model instantiated with this particular configuration of hyperparameters be $\mathscr{M}_\lambda$. The domain of a hyperparameter can be real-valued, integer-valued, binary, or categorical. For integer- and real-valued hyperparameters, the domains are typically bounded. It is also important to realize that the configuration space can

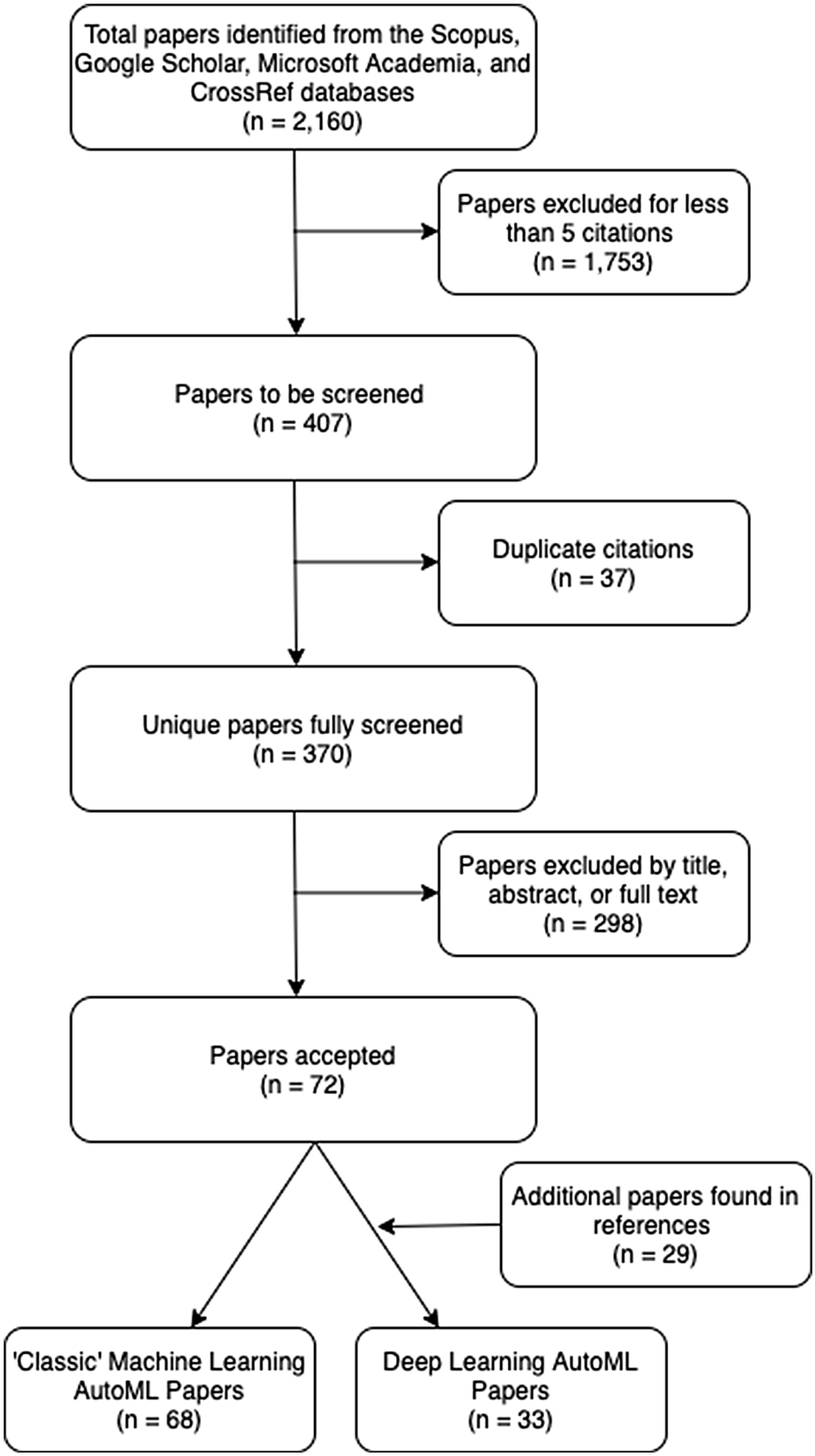

Fig. 2. The number of papers included/excluded at each stage in the screening phase of this literature review.

contain conditionality if a given hyperparameter is only relevant if another hyperparameter takes on a certain value. Given some dataset $\mathscr{D}$, the goal is to find

$$\lambda^{\star} = \underset{\lambda}{\text{argmin}} \mathrm{E}[V(\mathscr{L}, \mathscr{M}_{\lambda}, \mathscr{D}_{train}, \mathscr{D}_{valid})]$$

where $V(\mathscr{L}, \mathscr{M}_{\lambda}, \mathscr{D}_{train}, \mathscr{D}_{valid})$ measures the loss of the model

Table 1
Summary of the different automated feature engineering tools discussed in this section.

Method	Feature Engineering Technique	Citation Count
Deep Feature Synthesis [30]	Expand-Reduce	141
ExploreKit [31]	Expand-Reudce	53
One Button Machine [32]	Expand-Reduce	32
AutoLearn [33]	Expand-Reduce	16
GP Feature Construction [36]	Genetic Programming	68
Cognito [37]	Hiearchrial Greedy Search	38
RLFE [38]	Reinforcement Learning	21
LFE [39]	Meta-Learning	34

generated by algorithm $\mathscr{M}$ with hyperparameters λ on the training data $\mathscr{D}_{train}$ and validated on validation data $\mathscr{D}_{valid}$.

The simplest and most naive hyperparameter optimization strategies make no assumptions about the search space. Grid search is the simplest way to perform hyperparameter optimization, as it is a brute-force method in which the user specifies a finite set of values for each hyperparameter, and then evaluates the Cartesian product of these sets. This algorithm clearly suffers from the curse of dimensionality, as the search space grows exponentially with the size of the configuration space, and therefore it is often not a suitable choice given its large time requirements. A simple alternative to grid search is random search. Random search relies on sampling hyperparameter configurations from a user-specified set of hyperparameter values until a certain budget for the search is exhausted. Despite the fact that random search does not search as many configurations as grid search, it has still been shown to perform empirically better than grid search [43]. While grid search and random search are simple techniques that can serve as a useful baseline, neither of these methods make use of past performance evaluations and are therefore inefficient at exploring the search space.

Another class of hyperparameter optimization methods are "optimization from samples" methods [44], which are guided searches that iteratively generate new configurations based on the previous performance of prior configurations. Two popular examples of these types of methods are particle swarm optimization (PSO) [45] and evolutionary algorithms [46], both of which are inspired by biological behaviors. PSO is inspired by how biological communities interact at both the individual and the social level. PSO works by updating the configuration space at each iteration by moving the solution towards the best individual configurations and searching the neighboring configurations in later iterations. In contrast, evolutionary algorithms are inspired by biological evolution, and work by maintaining a population (configuration space) and improves the population by applying mutations (small perturbations) and crossover (combining individual solutions) to obtain a "generation" of better configurations. One of the best implementations of these population-based methods is the covariance matrix adaption evolution strategy (CMA-ES) [47], which samples configurations from a multivariate Gaussian distribution whose mean and covariance are updated in each generation.

In recent years, Bayesian optimization has emerged as the state-of-the-art optimization framework for AutoML systems. Bayesian optimization is a probabilistic, iterative algorithm with two main components: a surrogate model and an acquisition function. Bayesian optimization builds a probabilistic surrogate model, usually in the form of a Gaussian process [42,48] or a tree-based model [49,50], which is used to map the different hyperparameter configurations to their performance with some measure of uncertainty. Using this surrogate model, an acquisition function is then defined to determine the potential utility of a given configuration, and therefore balance exploration and exploitation during the search process. Bayesian optimization has a strong theoretical justification, and has been shown to work very well in practice [42,51–54] making it the most widely used method for optimizing hyperparameters. Several different open source implementations of Bayesian optimization exist, including Hyperopt [55], an unofficial implementation of Google Vizier [56], SMAC [50], Spearmint [57], Hyperas [58], and Talos [59]. For those interested in a more detailed explanation of Bayesian optimization, we refer to [60].

While much of the research done on hyperparameter optimization is impressive, most of these methods are still limited in efficiency within the context of large biomedical data environments. The amount of time required to search for optimal hyperparameters grows quite rapidly as the configuration space, dimensionality, and number of data points grows. In order to overcome some of these limitations, Zeng and Luo [21] propose an implementation of Bayesian optimization that uses progressive sampling [61] with the goal of building a tool to enable healthcare researchers to perform machine learning on their own. Similarly, the BOHB algorithm [62] has combined the principles of Bayesian optimization and Hyperband [63], a bandit search strategy that employs successive halving [64], in an attempt to build a more robust and efficient hyperparameter optimizer that works at scale. For those interested in learning more about the limitations of hyperparameter optimization in healthcare, we refer to [65]. Finally, for anyone interested in a more detailed look at the intricacies of hyperparameter optimization, we refer to [66].

5. Pipeline optimizers

The previous two sections only discussed methods that attempt to handle one component of the machine learning pipeline. However, in order for a machine learning system to be truly usable "off-the-shelf" by a non-expert, there is a need for AutoML systems that can be used to handle a variety of different tasks. In this section, we will consider these systems and refer to them as pipeline optimizers. Each pipeline optimizer performs one or more tasks in order to help automate the machine learning process.

The first pipeline optimizer we will consider is Auto-WEKA [67], an AutoML system based on the popular machine learning and data mining platform, WEKA [68]. Auto-WEKA was the first AutoML system that considered the problem of simultaneously selecting a machine learning algorithm and optimizing its hyperparameters; a problem which the creators dubbed the combined algorithm selection and hyperparameter optimization (CASH) problem. The CASH problem can be viewed as a single hierarchical hyperparameter optimization problem, where the choice of algorithm is itself a hyperparameter. The CASH problem can be more formally defined as follows.

First, let us consider the model selection problem. Let us consider a set of machine learning algorithms, $\mathscr{M}$, and some limited amount of training data $\mathscr{D} = [(x_1, y_1), ..., (x_n, y_n)]$. Our goal is to find the algorithm $\mathscr{M}^\star \in \mathscr{M}$ that gives us optimal performance. That is, we want to find

$$\mathscr{M}^\star \in \underset{\mathscr{M}}{\operatorname{argmin}} \frac{1}{k} \sum_{i=1}^{k} \mathscr{L}(\mathscr{M}, \mathscr{D}^{(i)}_{train}, \mathscr{D}^{(i)}_{valid})$$

where $\mathscr{L}(\mathscr{M}, \mathscr{D}^{(i)}_{train}, \mathscr{D}^{(i)}_{valid})$ is the loss achieved by $\mathscr{M}$ when trained on $\mathscr{D}^{(i)}_{train}$ and evaluated on $\mathscr{D}^{(i)}_{valid}$. This approach uses k-fold cross-validation [69] for assessing model performance. Using this formulation of the model selection problem, and the formulation of the hyperparameter optimization problem given in section II, we can define the CASH problem formally as follows.

Given a set of algorithms $\mathscr{M} = [M^{(1)}, ..., M^{(k)}]$ with associated hyperparameters spaces $\Lambda^{(1)}, ..., \Lambda^{(k)}$, the CASH problem is defined as

$$\mathscr{M}^\star_{\lambda^\star} \in \underset{\mathscr{M}_\lambda}{\operatorname{argmin}} \frac{1}{k} \sum_{i=1}^{k} \mathscr{L}(\mathscr{M}_\lambda, \mathscr{D}^{(i)}_{train}, \mathscr{D}^{(i)}_{valid})$$

Given this formulation, the authors were able to exploit Bayesian optimization methods to obtain high quality results in a reasonable amount of time. The Auto-WEKA platform utilizes the SMAC [50] optimization algorithm and solves the CASH problem using the learners and feature selectors implemented in the WEKA platform. Extensive empirical experiments on 21 prominent datasets showed that Auto-

WEKA often outperformed standard algorithm selection and hyperparameter optimization methods, especially on large datasets. Since its original release, the creators have made considerable improvements to the system and have dubbed the newer version as Auto-WEKA 2.0 [70]. Auto-WEKA 2.0 now supports regression problems (not just classification), optimization on all metrics available in WEKA, and is now fully integrated within the WEKA ecosystem rather than being a standalone piece of software.

The next pipeline optimizer we will consider is Auto-sklearn [71], which is often considered the state of the art for AutoML systems. The Auto-sklearn platform is based on the popular Python machine learning library scikit-learn [72]. Auto-sklearn also attempts to solve the CASH problem defined previously by the Auto-WEKA paper, and contains two improvements to the previous AutoML approach. The first improvement is a meta-learning step that is meant to warmstart the Bayesian optimization procedure [73] and therefore create a boost in efficiency. This meta-learning approach works by first evaluating a set of meta-features for 140 different datasets in the OpenML repository [74] (i.e. number of data points, number of features, data skewness, etc.) and then applying Bayesian optimization to determine and store a machine learning pipeline with strong empirical performance for each dataset. Then, when given a new dataset, the algorithm computes its meta-features, ranks all the other datasets by L_1 distance to $\mathscr{D}$ in the meta-feature space, and selects the stored machine learning pipelines for the $k = 25$ nearest datasets for evaluation before starting Bayesian optimization. The second improvement made by Auto-sklearn was the automated ensemble construction of models evaluated during optimization. While Bayesian optimization is data-efficient, it is also a wasteful procedure given that all the models it trains during the course of a given search are typically lost, including models that perform almost as well as the best performer. Rather than discarding these models, Auto-sklearn stores them and uses a post-processing method to construct an ensemble out of them. This ensemble construction avoids having to stick to one pipeline configuration and is therefore more robust and less prone to overfitting. It can also improve performance given the well-known tendency for ensembles to outperform individual models [75]. In empirical experiments, the Auto-sklearn system performed better than or equally as well as the Auto-WEKA system in 86 % of cases and also won first place in the previously mentioned ChaLearn AutoML challenge [76]. Despite its good performance, the original Auto-sklearn platform is limited to handling datasets of relatively modest size and has since been extended to handle larger datasets. The newest release of the system, known as PosSH Auto-sklearn [77], makes use of the BOHB algorithm described in section II, and has created a large increase in speed for the Auto-sklearn system. This PoSH Auto-skelarn system won the second iteration of the ChaLearn AutoML challenge. Nonetheless, Auto-sklearn and Auto-WEKA are still not well equipped to handle large clinical datasets, and we will elaborate on this problem in section VI.

A third popular pipeline optimizer is one that was originally developed to automate biomedical data science, but which has now been implemented to handle any machine learning task. This platform, known as the Tree-based Pipeline Optimization Tool (TPOT) [78], is an open source genetic programming-based AutoML system that is meant to handle feature preprocessing, model selection, and hyperparameter optimization tasks for a given machine learning problem. TPOT is a wrapper for scikit-learn, the Python machine learning library; therefore every machine learning operator corresponds to a machine learning algorithm that is present in that library. The TPOT system specifically uses supervised classification operators (i.e. Random Forest, KNN, Logistic Regression, etc.), feature preprocessing operators (i.e. Scalers, PCA, Polynomial Featurization, etc.), and feature selection operators (i.e. Variance Thresholders, RFE, etc.). In order to combine these operators into a full-fledged pipeline, they are all treated as genetic programming primitives, and genetic programming trees are constructed from them. To automatically generate and optimize these pipelines, the authors used a genetic programming algorithm described by [79], which is implemented in the Python package DEAP [80]. In empirical experiments conducted with the TPOT system, TPOT frequently discovered pipelines that performed statistically significantly better than a baseline Random Forest model, but the authors provided no comparison to other AutoML systems. The TPOT system was also run with a guided search and a random search mechanism, and they found that the random search mechanism often achieved comparable accuracy to the guided search mechanism. TPOT also provides a more flexible machine learning pipeline than the original Auto-sklearn system, but this sometimes causes the pipelines to be overfit on the data. Despite some of these drawbacks, TPOT receives regular improvements, and is still one of the most popular AutoML systems used to date.

While Auto-WEKA, Auto-sklearn, and TPOT are the three dominant, open-source AutoML pipeline optimizers to date, other attempts at approaching the problem have emerged in the past few years. The first we will discuss is the TuPAQ system [81], which was first published a year after the Auto-WEKA system. While essentially attempting to solve the same CASH problem, the authors frame joint model selection and hyperparameter optimization problem as a query optimization problem of their search space. TuPAQ uses a bandit search for its optimization based of properties of the data and the user-defined computational budget and is built upon the MLbase architecture [82]. Next, we have Auto-Tuned Models (ATM) [83], which is meant to be a distributed, collaborative, and scalable system for AutoML. The ATM platform is meant to allow data scientists to simply upload a dataset, select a desired machine learning method, and choose a hyperparameter range to search over. ATM then uses either a hybrid Bayesian/bandit optimization system or a model recommender system that uses meta-learning to optimize the pipeline. This system comes from the same authors as the DSM, which was mentioned in section III, and thus ATM also employs a feature engineering step. ATM is meant to be an automated feature engineer and a hyperparameter optimizer, which was demonstrated to match or exceed regular human performance on hundreds of different datasets, and take approximately $1/1000^{\text{th}}$ of the time. Around the same time as the ATM system was published, a different technique known as “Automatic Frankensteining” [84] emerged to solve the CASH problem using ensemble learning. Similar to the Auto-sklearn approach, the Automatic Frankensteining framework builds and selects well-optimized models, and then ensembles them to further boost prediction performance, as well as reduce the input space for the subsequent model selection component. When compared to Auto-WEKA and Auto-Sklearn on 80 different classification datasets from the UCI Machine Learning repository [85], this framework was able to outperform its competitors on the majority of datasets in the same CPU time, further demonstrating the usefulness of ensemble learning.

In the last year alone, several different approaches to the AutoML pipeline optimization problem have been developed. ML-Plan [86] uses hierarchical task networks (HTNs) [87], which have been used as an AI planning tool in the past [88]. ML-Plan encodes an HTN problem that divides the AutoML problem into two phases: algorithm selection and algorithm configuration. Their preliminary results show that ML-Plan is somewhat competitive with Auto-WEKA, but less so with Auto-sklearn. The authors are currently working to simultaneously optimize over pipelines with algorithms from the WEKA and scikit-learn libraries [89]. Autostacker [90], which was inspired by the ensemble learning technique known as stacking [91], uses an evolutionary algorithmic approach to perform hyperparameter optimization over hierarchical stacked machine learning models. This makes it similar to the TPOT approach, but it uses ensemble learning. Autostacker does not perform any data preprocessing or feature selection/engineering, yet it still performs competitively with TPOT, and is much faster. AlphaD3M [92] takes a reinforcement learning approach to the pipeline optimization problem by representing the model discovery process as a single-player game in which the “player” iteratively builds a pipeline by inserting, deleting, or replacing pipeline parts. This inherently makes the resulting pipeline incredibly interpretable, as it includes all the actions

Table 2
Summary of the different AutoML Pipeline optimizers discussed in this section.

Method	Optimization Algorithm	Data Pre-Processing	Feature Engineering	Model Selection	Hyperparameter Optimization	Ensemble Learning	Meta-Learning	Citation Count
Auto-WEKA [67,70]	Bayesian Optimization (SMAC)	✓		✓	✓			703
Auto-Sklearn [71,77]	Joint Bayesian Optimization and Bandit Search (BOHB)	✓		✓	✓	✓	✓	542
TPOT [78]	Evolutionary Algorithm	✓	✓	✓	✓			84
TuPAQ [81]	Bandit Search			✓	✓			94
ATM [83]	Joint Bayesian Optimization and Bandit Search		✓		✓		✓	29
Automatic Frankensteining [84]	Bayesian Optimization			✓	✓	✓		12
ML-Plan [86]	Hierarchical Task Networks (HTN)	✓		✓	✓			24
Autostacker [90]	Evolutionary Algorithm			✓	✓	✓		18
AlphaD3M [92]	Reinforcement Learning/Monte Carlo Tree Search	✓		✓	✓			8
Collaborative Filtering [94]	Probabilistic Matrix Factorization	✓		✓	✓		✓	29

and decisions which led to final pipeline creation. The authors use a sequence modeling technique that employs deep neural networks and Monte Carlo tree searches (MCTS), similarly to [93], to solve the optimization problem. AlphaD3M performs competitively on regression and classification problems from the OpenML repository, and computation times are an order of magnitude faster than other AutoML systems. One last notable approach uses probabilistic matrix factorization to tackle the AutoML problem [94] by modeling the problem of predicting machine learning pipeline performance as a collaborative filtering problem [95], which is frequently used in recommender systems. The empirical results indicate that this strategy can outperform Auto-sklearn in a majority of cases. These pipeline optimization methods are all summarized in the table below based on their capabilities.(Table 2)

6. Neural architecture search

In recent years, machine learning has been revolutionized by research in the field of deep learning [96]. Broadly speaking, deep learning is concerned with the construction of computational models, known as neural networks, that are composed of several different processing layers to learn representations of input data and map it to its associated output. Deep learning methods have significantly improved the state-of-the-art in perceptual learning tasks such as speech recognition [97,98], natural language processing [99–101], visual recognition [102,103], etc., as well as in tasks with large volumes of big data, such as genomics [104,105]. These deep learning algorithms use "neural networks" to find associations between inputs and outputs, and the basic structure of these networks are shown in Fig. 3.

A neural network is composed of three types of layers: an input layer, hidden layers, and an output layer. All layers are composed of nodes, which are sometimes called neurons. In the case of the hidden layer, the nodes are called hidden units. The input layer takes in some numerical representation of the data. The output layer produces a prediction. The hidden layers perform transformations on the data which are usually nonlinear. The outputs of each neuron are then fed into the subsequent layers, with different weights along the connections between the different neurons. For more detail on how these networks work, see [106]. The previously mentioned state-of-the-art neural networks are much larger than the network in Fig. 3, and often have much more complex network "architectures". Often thought of as a "black box" computational model, these networks can be incredibly complex, consisting of hundreds of millions of parameters to train, and their performance is highly dependent on their architecture and choice of hyperparameters [107–109]. The widespread success of these deep neural networks has created a need for architecture engineering, where data scientists are tasked with manually designing increasingly complex neural architectures. This has led to an increased interest among AutoML researchers to invest their time in the field of neural architecture search (NAS), which aims to find the best neural network architecture

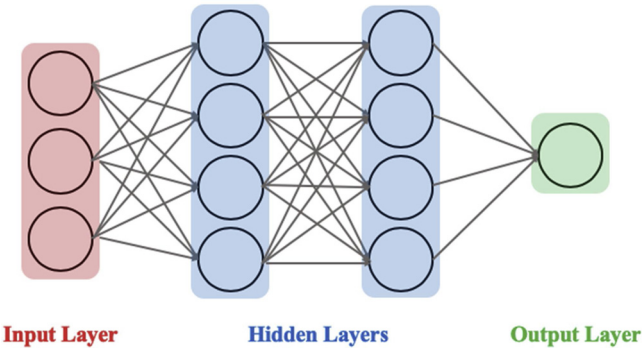

Fig. 3. The basic structure of a neural network, composed of input, hidden, and output layers.

for any given learning task and dataset. NAS is an important subfield of AutoML, with some overlap with hyperparameter optimization, and it will be the topic of discussion in this section.

As discussed in [110], NAS methods are best categorized by three factors: search space, search strategy, and performance estimation strategy. The search space refers to the potential neural architectures that can be represented by the NAS algorithm, whereas the search strategy refers to how this space is explored. Lastly, the performance estimation strategy refers to how the NAS algorithm evaluates a given architecture's performance on some task given some training dataset. All three of these components are non-trivial and come with important trade-offs to consider. The search space is susceptible to potential human bias if one tries to simplify the search with some sort of meta-learning technique, but the NAS may be slower if it does not do so. On the other hand, the search strategy must balance the notorious exploration-exploitation trade-off [111] as it aims to find the optimal architecture quickly without reaching a premature sub-optimal solution. Finally, performance estimation can be done simply by doing a standard training and validation of a neural architecture, but this is computationally expensive to carry out. We will now consider how different NAS algorithms address these issues.

6.1. Search space

The choice of search space significantly determines how difficult the optimization problem of NAS becomes, as optimization in this case is often non-continuous and usually high dimensional, given that more complex models tend to perform better. The simplest of search spaces is that of simple feed-forward neural networks [112], as demonstrated in Fig. 3, where the neural architecture $\mathscr{A}$ is written as a sequence of n layers, where the i^{th} layer L_i receives its inputs from layer $i-1$ and its output serves as the input for layer $i+1$. This search space is therefore parameterized by the number of layers n, the type of operation each layer can execute (i.e. nonlinear transformations, convolutions, pooling, etc.), and the hyperparameters associated with these operations (i.e. kernel and stride size for convolutional layers), thus making the search space a conditional one of non-fixed length. However, these simple feed-forward networks are often outperformed by more complex architectures. This has led to more recent NAS work [113–117] which focuses on incorporating modern neural design elements, such as skip connections [118], to search over the space of multi-branch networks [119]. Given that a lot of hand-crafted neural architectures use certain repeated motifs [115], some more recent NAS methods [114,120–123] propose searching for motifs, also known as cells or blocks, rather than for whole architectures. This motif search space significantly reduces the search space, while also achieving better performance than previous work [124] and lending itself more easily to transfer learning. Therefore, this has become the dominant method for establishing a search space in NAS.

6.2. Search strategy

As discussed previously in section II, Bayesian optimization has become the state-of-the-art for hyperparameter optimization, and early on that was the case for NAS as well. Bayesian optimization led to state-of-the-art computer vision architectures [125,126], and the first automatically tuned network to outperform human experts in a competition [127]. However, NAS really took off when Zoph and Le [124] framed NAS as a reinforcement learning (RL) [128] problem and obtained competitive results on the benchmark CIFAR-10 [129] and Penn Treebank [130] datasets. In this RL framework, the generation of a neural architecture is considered the agent's action and the action space is the search space. The agent's reward mechanism is then determined by the performance estimation strategy of the given NAS algorithm. Different RL approaches [114,121,124,131] differ in the agent's learning policy and how it is optimized.

An alternative to the popular RL approach is neuro-evolutionary approaches that use evolutionary algorithms for exploring the search space. NAS evolutionary algorithms [132–138] evolve a population of neural networks, and in every evolution step, at least one model from the population is sampled and serves as a parent to generate offspring by applying mutations to it. These mutations could be adding or removing a layer, altering hyperparameters, adding a skip connection, etc. After the offspring are generated, their fitness is evaluated using the performance estimation strategy, and they are added to the population of models. Different neuroevolutionary approaches vary by how they sample parent architectures, update populations, and generate offspring.

Given that RL and evolutionary algorithms are the dominant approaches for NAS, a recent study by Real et al. [139] compared the two methods along with a random search approach. Their results indicate that RL and evolutionary algorithms perform equally well in terms of prediction performance, but evolutionary approaches had a slight speed advantage and could find smaller models that perform just as well as more complex ones. While both methods outperformed random search, the margin was rather small with random search achieving a 4% test error on CIFAR-10, and RL and evolutionary approaches achieving 3.5 %.

While RL and neuroevolution are the standard for NAS, other search strategies have been suggested as well. Haifeng et al. recently proposed Auto-Keras [140], which is built upon the popular deep learning library Keras [141] and uses Bayesian optimization and network morphisms [142], a technique to alter the architecture of a network but keep its functionality, to speed up their search strategy [116]. and [143] also use network morphisms, but not Bayesian optimization. AdaNet [144] adaptively learns both the structure of the network and its weights. PNAS [123] uses sequential-model based optimization to search for structures in order of increasing complexity. DeepHyper [145] and [146] attempt to use Bayesian optimization to jointly learn network structure and hyperparameters to speed up the typical process of separating these two tasks. DeepArchitect [147] and [148] model the search space in a tree-structure and use Monte Carlo Tree Search. NASH [116] and GNAS [149] propose a greedy search method by moving in the direction of better performing architectures using a hill climbing approach. Finally, NAO [150] and DARTS [120] use a continuous relaxation of the search space constraints, thereby enabling the use of gradient-based optimization, which allows for the convex combination of multiple operations to be applied to the architecture at each iteration.

6.3. Performance estimation strategy

In order to guide these previously mentioned search strategies, a NAS algorithm needs a way to measure the performance of a given architecture that is being considered. While it would be very simple to train a given architecture on training data and evaluate its performance on a validation set, this is computationally expensive, sometimes taking thousands of GPU days [124,136]. In order to reduce this computational burden, several different techniques have been proposed. Some simple techniques include estimating performance with shorter training times [114,146], using a subset of the full training data [51], or using lower resolution images in computer vision tasks [151], but these techniques also introduce bias into the performance estimates. This bias isn't necessarily problematic as long as the relative rankings stay stable, but recent work suggests that this may not always be the case [146]. Another recent proposal for estimating performance is to use learning curve exploitation [126,152,153], which attempts to predict from the initial learning curves those architectures that will have a poor performance, and terminate those architectures to speed up the search [123]. also proposes performance prediction, but rather than using learning curves, the predictions are based on past performance of similar architectural/cell styles. The problem with performance

Table 3
The performance of several different NAS algorithms on the CIFAR-10 dataset.

NAS Algorithm	Search Space	Search Strategy	Performance Estimation Strategy	Number of Parameters	Search Time (GPU-days)	Test Error (%)
Large-scale Evolution [136]	Feed-Forward Networks	Evolutionary Algorithm	Naïve Training and Validation	5.4M	2600	5.4
EAS [143]	Feed-Forward Networks	Reinforcement Learning and Network Morphism	Short Training and Validation	23.4M	10	4.23
Hierarchical Evolution [138]	Cell Motifs	Evolutionary Algorithm	Training and Validation on proposed CNN Cell	15.7M	300	3.75
NAS v3 [124]	Multi-branched Networks	Reinforcement Learning	Naïve Training and Validation	37.4M	22400	3.65
PNAS [123]	Cell Motifs	Sequential Model-Based Optimization (SMBO)	Performance Prediction	3.2M	225	3.41
ENAS [121]	Cell Motifs	Reinforcement Learning	One Shot	4.6M	0.45	2.89
ResNet + Regularization [155]	Human Baseline			26.2M	–	2.86
DARTS [120]	Cell Motifs	Gradient-Based Optimization	Training and Validation on proposed CNN Cell	3.4M	4	2.83
NASNet-A [114]	Cell Motifs	Reinforcement Learning	Naïve Training and Validation	3.3M	2000	2.65
EENA [137]	Cell Motifs	Evolutionary Algorithm	Performance Prediction	8.5M	0.65	2.56
Path-Level EAS [115]	Cell Motifs	Reinforcement Learning	Short Training and Validation	14.3M	200	2.30
NAO [150]	Cell Motifs	Gradient-Based Optimization	Performance Prediction	128M	200	2.11

prediction is that in order to be useful, it requires good predictions in a very large search space with only a few evaluations. Another promising method to speed up performance estimation is to use a “one shot” method [117,120,121,154] that treats all neural architectures as subgraphs of a larger super-graph, and shares weights between architectures that have edges of this super-graph in common. Therefore, only the weights of a single model in the subgraph need to be trained, and any subsequent architectures in this same subgraph can be evaluated without any separate training. This method significantly speeds up performance estimation of architectures, but this method also suffers from severe bias. Despite this apparent flaw, it is has been shown to rank architectures reliably [154].

It is important to note that while NAS have achieved impressive performance on a variety of different tasks, it fails to provide any insights on why specific architectures work well. It has also been difficult to compare the different methods for NAS, as measuring architecture performance depends on a lot of factors other than the architecture itself. However, the results of several different NAS algorithms on the CIFAR-10 dataset is provided in Table 3. For those seeking a more detailed review of NAS, please see [110].

7. Automated machine learning in healthcare

Despite the growing research in the field of AutoML, there has been little work done in applying these techniques to the healthcare field despite demonstrated need [65]. There are several key challenges to applying machine learning in the healthcare space that make it very difficult to deploy AutoML solutions. An important challenge in any machine learning problem is assembling a high-quality, representative, and diverse dataset. Ideally, the machine learning model would be trained with data that exactly matches the format and quality of data that would be used at a later point. In a clinical context, this is often data in the electronic health records (EHR) format, which is problematic because EHR data is known to be unreliable and prone to unwanted variability [156,157]. These problems exist mainly because EHRs were originally designed for billing and coding and not necessarily for analytics nor to actively improve quality of care [158]. Furthermore, many health systems report using EHR systems that are only partially interoperable [158], making it difficult to obtain full patient histories. Not only is it difficult to apply any of the automated feature engineering techniques discussed in this paper on EHR data, but training machine learning models on this highly noisy data source may lead to undesirable results anyways.

While the lack of high-quality data is one potential impediment to deploying an AutoML system, a much bigger issue is the lack of transparency in these black-box AutoML systems. This lack of transparency in small decisions made by the system, such as what model configurations have been searched, leads machine learning experts and novices alike to question the automatic results. If users do not trust the AutoML system they are attempting to use, they will hesitate to apply the results of AutoML in critical applications [159], especially in healthcare where interpretability and transparency of algorithms are crucial for a system to be adopted into a work-flow [160]. In order to combat this issue, Wang et al. have recently proposed the ATMSeer system [161] as an interactive visualization tool that allows users to more easily analyze and refine the search space for an AutoML system. ATMSeer offers visual summaries of the searched models to improve transparency, while also allowing users to modify the search space in real time to improve the controllability of AutoML systems. While ATMSeer was integrated with the ATM system [83] described earlier, the authors state that ATMSeer is algorithm agnostic and should be able to integrate with a variety of AutoML frameworks. This is one example of a key tool that can be used to overcome some of the current limitations of AutoML.

Another potential reason for the lack of AutoML solutions in the healthcare space is that the current methods for machine learning pipeline optimization are inefficient for the type of large datasets that are

so common in the biomedical environment. In order to address some of these issues, Luo has proposed two software systems, MLBCD [162] and PredicT-ML [163], that are designed to automate machine learning for big clinical data. These systems are meant to specifically address efficiency issues [21], as well as common properties of clinical data in EHRs, such as temporal aggregation of clinical variables or processing data stored in the Entity-Attribute-Value (EAV) model [164], that sometimes limit the application of AutoML in healthcare. However, despite these systems' promising designs, they have not yet been clinically implemented, to our knowledge.

While the MLBCD and PredicT-ML systems have not been implemented, there have been some smaller use cases of AutoML in healthcare. Recently, AutoPrognosis [165] has emerged as a system for automatically constructing a machine learning pipeline that is tailored to the task of clinical prognosis. This system makes use of Bayesian optimization techniques to search the model and hyperparameter space, and even provides clinicians with association rules that link patients' clinical data to their predicted risk strata. The AutoPrognosis system also performs missing data imputation and feature preprocessing, and is able to handle different types of clinical data including longitudinal and time-to-event data. The creators of this system performed empirical experiments that showed that their AutoML technique outperformed traditional clinical scores, Auto-sklearn, Auto-WEKA, and TPOT in terms of Area Under the Curve (AUC) for prognostic modeling. Recently, the AutoPrognosis system has been shown to improve the accuracy of cardiovascular disease risk prediction [166], as well as cystic fibrosis prognostication [167], compared to scoring systems based on conventional risk factors. While the creators of AutoPrognosis created their own AutoML system, Orlenko et al. used the TPOT system for clinical metabolic profiling of patients exposed to metformin monotherapy [168]. Both of these examples point to the potential for AutoML to become a critical tool for helping healthcare practitioners and researchers turn large clinical data into actionable insights that help improve the quality of care.

8. Conclusion

Automated machine learning is an emerging research field within computer science that has the potential to help non-experts use machine learning off-the-shelf. We have reviewed the literature on a wide array of AutoML techniques, including hyperparameter optimization, automated feature engineering, pipeline optimization, and neural architecture search. While the ChaLearn AutoML Challenges have been able to benchmark a handful of AutoML pipeline optimizers on standardized hardware and specific machine learning tasks, there is still an opportunity for other benchmarking platforms, such as Kaggle, to step in and better evaluate other existing tools/systems in a standardized manner in order to guide users when selecting which tools to use. It is worth mentioning that while our review has focused on open source AutoML tools, there are several industry products as well, including Google Cloud's AutoML system [169], Amazon SageMaker [170] and Amazon Comprehend [171], Microsoft Azure AutoML [172], H_20 Driverless AI [173], BigML's OptiML [174], and DataRobot [175]. There has been some work that attempts to benchmark the open-source auto-sklearn and TPOT systems against H20′s product and the results favored auto-sklearn for classification and TPOT for regression [176]. However, as mentioned previously, more work needs to be done in this regard. We identify some limitations of current AutoML approaches, in particular how they fail to handle the size and variety of data within biomedical environments. While there have already been some use cases of AutoML in the healthcare field, more work needs to be done for there to be a widespread adoption of AutoML in healthcare. This survey should act as a basic guide for healthcare researchers interested in applying data science techniques to their domain of interest.

Declaration of Competing Interest

All authors declare that they have no significant competing financial, professional, or personal interests that might have influenced the performance or presentation of the work described in this manuscript.

Acknoledgement

Authors would like to thank Edward Moseley, Gregory Antell, Chih-Ying Deng, Jacob Rosenthal, Clyde Bango, and Jason Johnson for their support and suggestions that improved this work.

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at https://doi.org/10.1016/j.artmed.2020.101822.

References

[1] Luo J, et al. Big data application in biomedical research and health care: a literature review. Biomed Inform Insights 2016;8. p. BII. S31559.
[2] Toga AW, et al. Big biomedical data as the key resource for discovery science. J Am Med Inform Assoc 2015;22(6):1126–31.
[3] Murdoch TB, Detsky AS. The inevitable application of big data to health care. Jama 2013;309(13):1351–2.
[4] Brown, N. Healthcare Data Growth: An Exponential Problem. 2015 5/22/2019; Available from: https://www.nextech.com/blog/healthcare-data-growth-an-exponential-problem.
[5] Lundberg SM, et al. Explainable machine-learning predictions for the prevention of hypoxaemia during surgery. Nat Biomed Eng 2018;2(10):749.
[6] Saria S, Koller D, Penn A. Learning individual and population level traits from clinical temporal data. Proceedings of Neural Information Processing Systems. 2010.
[7] Marella WM, Sparnon E, Finley E. Screening electronic health Record–Related patient safety reports using machine learning. J Patient Saf 2017;13(1):31–6.
[8] Kuo C-C, et al. Automation of the kidney function prediction and classification through ultrasound-based kidney imaging using deep learning. Npj Digit Med 2019;2(1):29.
[9] Rumsfeld JS, Joynt KE, Maddox TM. Big data analytics to improve cardiovascular care: promise and challenges. Nat Rev Cardiol 2016;13(6):350.
[10] Liang H, et al. Evaluation and accurate diagnoses of pediatric diseases using artificial intelligence. Nat Med 2019:1.
[11] Bates DW, et al. Big data in health care: using analytics to identify and manage high-risk and high-cost patients. Health Aff 2014;33(7):1123–31.
[12] Özdemir A, Barshan B. Detecting falls with wearable sensors using machine learning techniques. Sensors 2014;14(6):10691–708.
[13] Lo-Ciganic W-H, et al. Using machine learning to examine medication adherence thresholds and risk of hospitalization. Med Care 2015;53(8):720.
[14] Escobar GJ, et al. Piloting electronic medical record–based early detection of inpatient deterioration in community hospitals. J Hosp Med 2016;11:S18–24.
[15] Rajpurkar P, et al. Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. arXiv preprint arXiv:1711.05225. 2017.
[16] Ardila D, et al. End-to-end lung cancer screening with three-dimensional deep learning on low-dose chest computed tomography. Nat Med 2019.
[17] Kose I, Gokturk M, Kilic K. An interactive machine-learning-based electronic fraud and abuse detection system in healthcare insurance. Appl Soft Comput 2015;36:283–99.
[18] Rajkomar A, Dean J, Kohane I. Machine learning *in medicine*. N Engl J Med 2019;380(14):1347–58.
[19] Beam AL, Kohane IS. Big data and machine learning in health care. Jama 2018;319(13):1317–8.
[20] Weintraub WS, Fahed AC, Rumsfeld JS. Translational medicine in the era of big data and machine learning. Circ Res 2018;123(11):1202–4.
[21] Zeng X, Luo G. Progressive sampling-based Bayesian optimization for efficient and automatic machine learning model selection. Health Inf Sci Syst 2017;5(1):2.
[22] Wolpert DH, Macready WG. No free lunch theorems for optimization. Ieee Trans Evol Comput 1997;1(1):67–82.
[23] Auffray C, et al. Making sense of big data in health research: towards an EU action plan. Genome Med 2016;8(1):71.
[24] Guyon I, et al. Analysis of the AutoML Challenge series. 2017. p. 2015–8.
[25] Quanming Y, et al. Taking Human Uut of Learning Applications: A Survey on Automated Machine Learning. arXiv preprint arXiv:1810.13306. 2018.
[26] Domingos PM. A few useful things to know about machine learning. Commun ACM 2012;55(10):78–87.
[27] Bengio Y, Courville A, Vincent P. Representation learning: a review and new perspectives. IEEE Trans Pattern Anal Mach Intell 2013;35(8):1798–828.
[28] Miotto R, et al. Deep patient: an unsupervised representation to predict the future of patients from the electronic health records. Sci Rep 2016;6:26094.
[29] Rajkomar A, et al. Scalable and accurate deep learning with electronic health

records. Npj Digit Med 2018;1(1):18.
[30] Kanter JM, Veeramachaneni K. Deep feature synthesis: towards automating data science endeavors. 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA). 2015.
[31] Katz G, Shin ECR, Song D. Explorekit: automatic feature generation and selection. 2016 IEEE 16th International Conference on Data Mining (ICDM). 2016.
[32] Lam HT, et al. One button machine for automating feature engineering in relational databases. arXiv preprint arXiv:1706.00327. 2017.
[33] Kaul A, Maheshwary S, Pudi V. Autolearn—Automated feature generation and selection. 2017 IEEE International Conference on Data Mining (ICDM). 2017.
[34] FeatureLabs. Featuretools. Available from:https://github.com/featuretools/featuretools.
[35] Smith MJ, Wedge R, Veeramachaneni K. FeatureHub: towards collaborative data science. 2017 IEEE International Conference on Data Science and Advanced Analytics (DSAA). 2017.
[36] Tran B, Xue B, Zhang M. Genetic programming for feature construction and selection in classification on high-dimensional data. Memetic Comput 2016;8(1):3–15.
[37] Khurana U, et al. Cognito: automated feature engineering for supervised learning. 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW). 2016.
[38] Khurana U, Samulowitz H, Turaga D. Feature engineering for predictive modeling using reinforcement learning. Thirty-Second AAAI Conference on Artificial Intelligence. 2018.
[39] Nargesian F, et al. Learning feature engineering for classification. Ijcai. 2017.
[40] Hoos H, Ca U, Leyton-Brown K. An efficient approach for assessing hyperparameter importance. International Conference on Machine Learning. 2014.
[41] Komer B, Bergstra J, Eliasmith C. Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. ICML workshop on AutoML. Citeseer; 2014.
[42] Snoek J, Larochelle H, Adams RP. Practical bayesian optimization of machine learning algorithms. Advances in neural information processing systems. 2012.
[43] Bergstra J, Bengio Y. Random search for hyper-parameter optimization. J Mach Learn Res 2012;13:281–305. Feburary.
[44] Conn AR, Scheinberg K, Vicente LN. Introduction to derivative-free optimization Vol. 8. Siam; 2009.
[45] Escalante HJ, Montes M, Sucar LE. Particle swarm model selection. J Mach Learn Res 2009;10:405–40. Feburary.
[46] Back T. Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford university press; 1996.
[47] Hansen N. The CMA evolution strategy: A tutorial. arXiv preprint arXiv:1604.00772. 2016.
[48] Wistuba M, Schilling N, Schmidt-Thieme L. Scalable Gaussian process-based transfer surrogates for hyperparameter optimization. Mach Learn 2018;107(1):43–78.
[49] Bergstra JS, et al. Algorithms for hyper-parameter optimization. Advances in neural information processing systems. 2011.
[50] Hutter F, Hoos HH, Leyton-Brown K. Sequential model-based optimization for general algorithm configuration. International Conference on Learning and Intelligent Optimization. Springer; 2011.
[51] Klein A, et al. Fast bayesian optimization of machine learning hyperparameters on large datasets. arXiv preprint arXiv:1605.07079. 2016.
[52] Dahl GE, Sainath TN, Hinton GE. Improving deep neural networks for LVCSR using rectified linear units and dropout. 2013 IEEE international conference on acoustics, speech and signal processing. 2013.
[53] Melis G, Dyer C, Blunsom P. On the state of the art of evaluation in neural language models. arXiv preprint arXiv:1707.05589. 2017.
[54] Snoek J, et al. Scalable bayesian optimization using deep neural networks. International conference on machine learning 2015.
[55] Bergstra J, Yamins D, Cox DD. Hyperopt: a python library for optimizing the hyperparameters of machine learning algorithms. Proceedings of the 12th Python in science conference. 2013.
[56] Golovin D, et al. Google vizier: A service for black-box optimization. Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2017.
[57] Group, H.I.P.S. Spearmint. Available from: https://github.com/HIPS/Spearmint.
[58] Pumperia, M. Hyperas. Available from: https://github.com/maxpumperla/hyperas.
[59] Autonomio. Talos. Available from: https://github.com/autonomio/talos.
[60] Shahriari B, et al. Taking the human out of the loop: a review of bayesian optimization. Proc Ieee 2016;104(1):148–75.
[61] Provost F, Jensen D, Oates T. Efficient progressive sampling. Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 1999.
[62] Falkner S, Klein A, Hutter F. Bohb: Robust and efficient hyperparameter optimization at scale. arXiv preprint arXiv:1807.01774. 2018.
[63] Li L, et al. Hyperband: A novel bandit-based approach to hyperparameter optimization. arXiv preprint arXiv:1603.06560. 2016.
[64] Jamieson K, Talwalkar A. Non-stochastic best arm identification and hyperparameter optimization. Artificial Intelligence and Statistics. 2016.
[65] Luo G. A review of automatic selection methods for machine learning algorithms and hyper-parameter values. Netw Model Anal Health Inform Bioinform 2016;5(1):18.
[66] Feurer M, Hutter F. Hyperparameter optimization. Automatic Machine Learning: Methods, Systems, Challenges. Springer; 2018. p. 3–38.
[67] Thornton C, et al. Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2013.
[68] Holmes G, Donkin A, Witten IH. Weka: A machine learning workbench. 1994.
[69] Kohavi R. A study of cross-validation and bootstrap for accuracy estimation and model selection. Ijcai. 1995. Montreal, Canada.
[70] Kotthoff L, et al. Auto-WEKA 2.0: automatic model selection and hyperparameter optimization in WEKA. J Mach Learn Res 2017;18(1):826–30.
[71] Feurer M, et al. Efficient and robust automated machine learning. Advances in neural information processing systems. 2015.
[72] Pedregosa F, et al. Scikit-learn: machine learning in Python. J Mach Learn Res 2011;12:2825–30. October.
[73] Feurer M, Springenberg JT, Hutter F. Initializing bayesian hyperparameter optimization via meta-learning. Twenty-Ninth AAAI Conference on Artificial Intelligence 2015.
[74] Vanschoren J, et al. OpenML: networked science in machine learning. Acm Sigkdd Explor Newsl 2014;15(2):49–60.
[75] Lacoste A, et al. Agnostic bayesian learning of ensembles. International Conference on Machine Learning 2014.
[76] Guyon I, et al. Design of the 2015 ChaLearn AutoML challenge. 2015 International Joint Conference on Neural Networks (IJCNN). 2015.
[77] Feurer M, et al. Practical automated machine learning for the automl challenge 2018. International Workshop on Automatic Machine Learning at ICML 2018.
[78] Olson RS, et al. Automating biomedical data science through tree-based pipeline optimization. European Conference on the Applications of Evolutionary Computation. 2016.
[79] Banzhaf W, et al. Genetic programming: an introduction Vol. 1. San Francisco: Morgan Kaufmann; 1998.
[80] Fortin F-A, et al. DEAP: evolutionary algorithms made easy. J Mach Learn Res 2012;13:2171–5. July.
[81] Sparks ER, et al. Automating model search for large scale machine learning. Proceedings of the Sixth ACM Symposium on Cloud Computing. 2015.
[82] Kraska T, et al. MLbase: a distributed machine-learning system. Cidr. 2013.
[83] Swearingen T, et al. ATM: a distributed, collaborative, scalable system for automated machine learning. 2017 IEEE International Conference on Big Data (Big Data). 2017.
[84] Wistuba M, Schilling N, Schmidt-Thieme L. Automatic frankensteining: creating complex ensembles autonomously. in Proceedings of the 2017 SIAM International Conference on Data Mining. 2017.
[85] Asuncion A, Newman D. UCI machine learning repository. 2007.
[86] Mohr F, Wever M, Hüllermeier E. ML-Plan: automated machine learning via hierarchical planning. Mach Learn 2018;107(8–10):1495–515.
[87] Ghallab M, Nau D, Traverso P. Automated Planning: theory and practice. Elsevier; 2004.
[88] Nau DS, et al. SHOP2: an HTN planning system. J Artif Intell Res 2003;20:379–404.
[89] Mohr F, et al. Towards the automated composition of machine learning service. IEEE International Conference on Services Computing. 2018.
[90] Chen B, et al. Autostacker: a compositional evolutionary learning system. Proceedings of the Genetic and Evolutionary Computation Conference. 2018.
[91] Wolpert DH. Stacked generalization. Neural Netw 1992;5(2):241–59.
[92] Drori I, et al. AlphaD3M: machine learning pipeline synthesis. AutoML Workshop at ICML. 2018.
[93] Rakotoarison H, Sebag M. AutoML with Monte carlo tree search. Workshop AutoML 2018@ ICML/IJCAI-ECAI. 2018.
[94] Fusi N, Sheth R, Elibol M. Probabilistic matrix factorization for automated machine learning. Advances in Neural Information Processing Systems. 2018.
[95] Sarwar BM, et al. Item-based collaborative filtering recommendation algorithms. Www 2001;1:285–95.
[96] LeCun Y, Bengio Y, Hinton G. Deep learning. nature 2015;521(7553):436.
[97] Hinton G, et al. Deep neural networks for acoustic modeling in speech recognition. IEEE Signal Process Mag 2012;29.
[98] Graves A, Mohamed A-r, Hinton G. Speech recognition with deep recurrent neural networks. 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. 2013.
[99] Collobert R, et al. Natural language processing (almost) from scratch. J Mach Learn Res 2011;12:2493–537. August.
[100] Jean S, et al. On using very large target vocabulary for neural machine translation. arXiv preprint arXiv:1412.2007. 2014.
[101] Bordes A, Chopra S, Weston J. Question answering with subgraph embeddings. arXiv preprint arXiv:1406.3676. 2014.
[102] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems. 2012.
[103] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556. 2014.
[104] Alipanahi B, et al. Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning. Nat Biotechnol 2015;33(8):831.
[105] Asgari E, Mofrad MR. Continuous distributed representation of biological sequences for deep proteomics and genomics. PLoS One 2015;10(11):e0141287.
[106] Bishop CM. Neural networks for pattern recognition. Oxford university press; 1995.
[107] Yue-Hei Ng J, et al. Beyond short snippets: deep networks for video classification. Proceedings of the IEEE conference on computer vision and pattern recognition 2015.
[108] He K, et al. Identity mappings in deep residual networks. European conference on computer vision. 2016.
[109] Che Z, et al. Recurrent neural networks for multivariate time series with missing

values. Sci Rep 2018;8(1):6085.
[110] Elsken T, Metzen JH, Hutter F. Neural architecture search: a survey. J Mach Learn Res 2019;20(55):1–21.
[111] March JG. Exploration and exploitation in organizational learning. Organ Sci 1991;2(1):71–87.
[112] Svozil D, Kvasnicka V, Pospichal J. Introduction to multi-layer feed-forward neural networks. Chemom Intell Lab Syst 1997;39(1):43–62.
[113] Elsken T, Metzen JH, Hutter F. Efficient Multi-objective Neural Architecture Search via Lamarckian Evolution. arXiv preprint arXiv:1804.09081. 2018.
[114] Zoph B, et al. Learning transferable architectures for scalable image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2018.
[115] Cai H, et al. Path-level network transformation for efficient architecture search. arXiv preprint arXiv:1806.02639. 2018.
[116] Elsken T, Metzen J-H, Hutter F. Simple and efficient architecture search for convolutional neural networks. arXiv preprint arXiv:1711.04528. 2017.
[117] Brock A, et al. SMASH: one-shot model architecture search through hypernetworks. arXiv preprint arXiv:1708.05344. 2017.
[118] Drozdzal M, et al. The importance of skip connections in biomedical image segmentation. Deep Learning and Data Labeling for Medical Applications. Springer; 2016. p. 179–87.
[119] Yamashita T, et al. Multi-branch structure of layered neural networks. Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP'02. 2002.
[120] Liu H, Simonyan K, Yang Y. Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055. 2018.
[121] Pham H, et al. Efficient neural architecture search via parameter sharing. arXiv preprint arXiv:1802.03268. 2018.
[122] Zhong Z, et al. Practical block-wise neural network architecture generation. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2018.
[123] Liu C, et al. Progressive neural architecture search. Proceedings of the European Conference on Computer Vision (ECCV) 2018.
[124] Zoph B, Le QV. Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578. 2016.
[125] Bergstra J, Yamins D, Cox DD. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. 2013.
[126] Domhan T, Springenberg JT, Hutter F. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. Twenty-Fourth International Joint Conference on Artificial Intelligence 2015.
[127] Mendoza H, et al. Towards automatically-tuned neural networks. Workshop on Automatic Machine Learning. 2016.
[128] Sutton RS, Barto AG. Reinforcement learning: An introduction. MIT press; 2018.
[129] Krizhevsky A, Nair V, Hinton G. The CIFAR-10 dataset 55. 2014. online: http://www.cs. toronto.edu/kriz/cifar.html.
[130] Marcus M, Santorini B, Marcinkiewicz MA. Building a Large Annotated Corpus of English: The Penn Treebank. 1993.
[131] Baker B, et al. Designing neural network architectures using reinforcement learning. arXiv preprint arXiv:1611.02167. 2016.
[132] Stanley KO, et al. Designing neural networks through neuroevolution. Nature Machine Intelligence 2019;1(1):24–35.
[133] Liang J, et al. Evolutionary Neural AutoML for Deep Learning. arXiv preprint arXiv:1902.06827. 2019.
[134] Miikkulainen R, et al. Evolving deep neural networks. Artificial Intelligence in the Age of Neural Networks and Brain Computing. Elsevier; 2019. p. 293–312.
[135] Suganuma M, Shirakawa S, Nagao T. A genetic programming approach to designing convolutional neural network architectures. Proceedings of the Genetic and Evolutionary Computation Conference. 2017.
[136] Real E, et al. Large-scale evolution of image classifiers. Proceedings of the 34th International Conference on Machine Learning-Volume 70. 2017.
[137] Zhu H, et al. EENA: Efficient Evolution of Neural Architecture. arXiv preprint arXiv:1905.07320. 2019.
[138] Liu H, et al. Hierarchical representations for efficient architecture search. arXiv preprint arXiv:1711.00436. 2017.
[139] Real E, et al. Evolutionary algorithms and reinforcement learning: a comparative case study for architecture search. Proceedings of Machine Learning Research, ICML 2018 AutoML Workshop 2018.
[140] Jin H, Song Q, Hu X. Efficient neural architecture search with network morphism. arXiv preprint arXiv:1806.10282. 2018.
[141] Chollet F. Keras. 2015.
[142] Wei T, et al. Network morphism. International Conference on Machine Learning 2016.
[143] Cai H, et al. Efficient architecture search by network transformation. Thirty-Second AAAI Conference on Artificial Intelligence 2018.
[144] Cortes C, et al. Adanet: adaptive structural learning of artificial neural networks. Proceedings of the 34th International Conference on Machine Learning-Volume 70. 2017.
[145] Balaprakash P, et al. deepHyper: asynchronous hyperparameter search for deep neural networks. 2018 IEEE 25th International Conference on High Performance Computing (HiPC). 2018.
[146] Zela A, et al. Towards Automated Deep Learning: Efficient Joint Neural Architecture and Hyperparameter Search. arXiv preprint arXiv:1807.06906. 2018.
[147] Negrinho R, Gordon G. Deeparchitect: Automatically Designing and Training Deep Architectures. arXiv preprint arXiv:1704.08792. 2017.
[148] Wistuba M. Finding Competitive Network Architectures within a Day Using UCT. arXiv preprint arXiv:1712.07420. 2017.
[149] Huang S, et al. GNAS: A Greedy Neural Architecture Search Method for Multi-Attribute Learning. 2018 ACM Multimedia Conference on Multimedia Conference. 2018.
[150] Luo R, et al. Neural architecture optimization. Advances in Neural Information Processing Systems. 2018.
[151] Chrabaszcz P, Loshchilov I, Hutter F. A downsampled variant of imagenet as an alternative to the cifar datasets. arXiv preprint arXiv:1707.08819. 2017.
[152] Baker B, et al. Accelerating neural architecture search using performance prediction. arXiv preprint arXiv:1705.10823. 2017.
[153] Rawal A, Miikkulainen R. From nodes to networks: evolving recurrent neural networks. arXiv preprint arXiv:1803.04439. 2018.
[154] Bender G, et al. Understanding and simplifying one-shot architecture search. International Conference on Machine Learning 2018.
[155] Gastaldi X. Shake-shake regularization. arXiv preprint arXiv:1705.07485. 2017.
[156] Hersh WR, et al. Caveats for the use of operational electronic health record data in comparative effectiveness research. Med Care 2013;51(8 0 3):S30.
[157] Elmore JG, et al. Pathologists' diagnosis of invasive melanoma and melanocytic proliferations: observer accuracy and reproducibility study. bmj 2017;357:j2813.
[158] Polite BN, et al. State of Cancer care in America: reflections on an inaugural year. American Society of Clinical Oncology 2019.
[159] Luhmann N. Trust and power. John Wiley & Sons; 2018.
[160] Luo G. Automatically explaining machine learning prediction results: a demonstration on type 2 diabetes risk prediction. Health Inf Sci Syst 2016;4(1):2.
[161] Wang Q, et al. ATMSeer: increasing transparency and controllability in automated machine learning. arXiv preprint arXiv:1902.05009. 2019.
[162] Luo G. MLBCD: a machine learning tool for big clinical data. Health Inf Sci Syst 2015;3(1):3.
[163] Luo G. PredicT-ML: a tool for automating machine learning model building with big clinical data. Health Inf Sci Syst 2016;4(1):5.
[164] Dinu V, Nadkarni P. Guidelines for the effective use of entity–attribute–value modeling for biomedical databases. Int J Med Inform 2007;76(11–12):769–79.
[165] Alaa AM, van der Schaar M. Autoprognosis: automated clinical prognostic modeling via bayesian optimization with structured kernel learning. arXiv preprint arXiv:1802.07207. 2018.
[166] Alaa AM, et al. Cardiovascular disease risk prediction using automated machine learning: a prospective study of 423,604 UK Biobank participants. PLoS One 2019;14(5):e0213653.
[167] Alaa AM, van der Schaar M. Prognostication and risk factors for cystic fibrosis via automated machine learning. Sci Rep 2018;8(1):11242.
[168] Orlenko A, et al. Considerations for automated machine learning in clinical metabolic profiling: altered homocysteine plasma concentration associated wtih metformin exposure. Pac symp biocomput. World Scientific; 2017.
[169] Cloud AutoML. 5/20/2019]; Available from: https://cloud.google.com/automl/.
[170] Amazon SageMaker. 5/20/2019]; Available from: https://aws.amazon.com/sagemaker/.
[171] Amazon Comprehend 5/20/2019]; Available from: https://aws.amazon.com/comprehend/?nc=sn&loc=2&dn=1&exp=b.
[172] Mukunthu D. Announcing automated ML capability in azure machine learning. Microsoft Azure 2018.
[173] H2O Driverless AI. 5/20/2019]; Available from: https://www.h2o.ai/products/h2o-driverless-ai/.
[174] Jesus M, Release Big ML. Automatically find the optimal machine learning model with OptiML!. BigML 2018.
[175] Automated Machine Learning. 5/20/2019]; Available from: https://www.datarobot.com/platform/automated-machine-learning/.
[176] Balaji A, Allen A. Benchmarking Automatic Machine Learning Frameworks. arXiv preprint arXiv:1808.06492. 2018.