# Automated Machine Learning: State-of-The-Art and Open Challenges

Radwa Elshawi
radwa.elshawi@ut.ee
radwa.elshawi@ut.ee

Mohamed Maher
University of Tartu, Estonia
mohamed.maher@ut.ee

Sherif Sakr
University of Tartu, Estonia
sherif.sakr@ut.ee

## ABSTRACT

Nowadays, machine learning techniques and algorithms are employed in almost every application domain (e.g., financial applications, advertising, recommendation systems, user behavior analytics). In practice, they are playing a crucial role in harnessing the power of massive amounts of data which we are currently producing every day in our digital world. In general, the process of building a high-quality machine learning model is an iterative, complex and time-consuming process that involves trying different algorithms and techniques in addition to having a good experience with effectively tuning their hyper-parameters. In particular, conducting this process efficiently requires solid knowledge and experience with the various techniques that can be employed. With the continuous and vast increase of the amount of data in our digital world, it has been acknowledged that the number of knowledgeable data scientists can not scale to address these challenges. Thus, there was a crucial need for automating the process of building good machine learning models. In the last few years, several techniques and frameworks have been introduced to tackle the challenge of automating the process of `C`ombined `A`lgorithm `S`election and `H`yper-parameter tuning (CASH) in the machine learning domain. The main aim of these techniques is to reduce the role of human in the loop and fill the gap for non-expert machine learning users by playing the role of the domain expert.

In this paper, we present a comprehensive survey for the state-of-the-art efforts in tackling the CASH problem. In addition, we highlight the research work of automating the other steps of the full complex machine learning pipeline (AutoML) from data understanding till model deployment. Furthermore, we provide a comprehensive coverage for the various tools and frameworks that have been introduced in this domain. Finally, we discuss some of the research directions and open challenges that need to be addressed in order to achieve the vision and goals of the AutoML process.

## 1. INTRODUCTION

Due to the increasing success of machine learning techniques in several application domains, they have been attracting a lot of attention from the research and business communities. In general, the effectiveness of machine learning techniques mainly rests on the availability of massive datasets. Recently, we have been witnessing a continuous exponential growth in the size of data produced by various kinds of systems, devices and data sources. It has been reported that there are 2.5 quintillion bytes of data is being created everyday where 90% of stored data in the world, has been generated in the past two years only[1]. On the one hand, the more data that is available, the richer and the more robust the insights and the results that machine learning techniques can produce. Thus, in the Big Data Era, we are witnessing many leaps achieved by machine and deep learning techniques in a wide range of fields [135, 106]. On the other hand, this situation is raising a potential *data science crisis*, similar to the software crisis [41], due to the crucial need of having an increasing number of data scientists with strong knowledge and good experience so that they are able to keep up with harnessing the power of the massive amounts of data which are produced daily. In particular, it has been acknowledged that *data scientists can not scale*[2] and it is almost impossible to balance between the number of qualified data scientists and the required effort to manually analyze the increasingly growing sizes of available data. Thus, we are witnessing a growing focus and interest to support automating the process of building machine learning pipelines where the presence of a human in the loop can be dramatically reduced, or preferably eliminated.

In general, the process of building a high-quality machine learning model is an iterative, complex and time-consuming process that involves a number of steps (Figure 1). In particular, a data scientist is commonly *challenged* with a large number of choices where informed decisions need to be taken. For example, the data scientist needs to select among a wide range of possible algorithms including classification or regression techniques (e.g. Support Vector Machines, Neural Networks, Bayesian Models, Decision Trees, etc) in addition to tuning numerous hyper-parameters of the selected algorithm. In addition, the performance of the model can also be judged by various metrics (e.g., accuracy, sensitivity, specificity, F1-score). Naturally, the decisions of the

[1]Forbes: How Much Data Do We Create Every Day? May 21, 2018
[2]`https://hbr.org/2015/05/data-scientists-dont-scale`

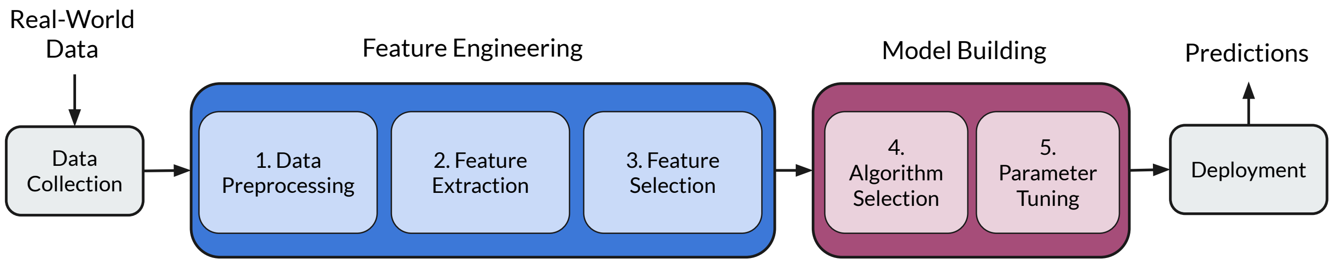

**Figure 1: Typical Supervised Machine Learning Pipeline.**

data scientist in each of these steps affect the performance and the quality of the developed model [122, 100, 95]. For instance, in `yeast dataset`[3], different parameter configurations of a Random Forest classifier result in different range of accuracy values, around 5%[4]. Also, using different classifier learning algorithms leads to widely different performance values, around 20% , for the fitted models on the same dataset. Although making such decisions require solid knowledge and expertise, in practice, increasingly, users of machine learning tools are often non-experts who require *off-the-shelf* solutions. Therefore, there has been a growing interest to *automate* and *democratize* the steps of building the machine learning pipelines.

In the last years, several techniques and frameworks have been introduced to tackle the challenge of automating the process of Combined Algorithm Selection and Hyper-parameter tuning (CASH) in the machine learning domain. These techniques have commonly formulated the problem as an optimization problem that can be solved by wide range of techniques [66, 39, 78]. In general, the *CASH* problem is described as follows:

Given a set of machine learning algorithms $\mathbf{A} = \{A^{(1)}, A^2, ...\}$, and a dataset $D$ divided into disjoint training $D_{train}$, and validation $D_{validation}$ sets. The goal is to find an algorithm $A^{(i)^*}$ where $A^{(i)} \in \mathbf{A}$ and $A^{(i)^*}$ is a tuned version of $A^{(i)}$ that achieves the highest generalization performance by training $A^{(i)}$ on $D_{train}$, and evaluating it on $D_{validation}$. In particular, the goal of any CASH optimization technique is defined as:

$$A^{(i)^*} \in \underset{A\ \epsilon\ \mathbf{A}}{argmin}\ L(A^{(i)}, D_{train}, D_{validation})$$

where L($A^{(i)}$, $D_{train}$, $D_{validation}$) is the loss function (e.g: error rate, false positives, etc). In practice, one constraint for CASH optimization techniques is the *time budget*. In particular, the aim of the optimization algorithm is to select and tune a machine learning algorithm that can achieve (near)-optimal performance in terms of the user-defined evaluation metric (e.g., accuracy, sensitivity, specificity, F1-score) within the user-defined *time budget* for the search process (Figure 2).

[3]`https://www.openml.org/d/40597`
[4]`https://www.openml.org/t/2073`

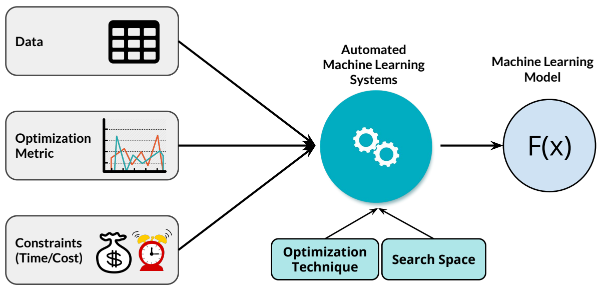

**Figure 2: The general Workflow of the AutoML process.**

In this paper, we present a comprehensive survey for the state-of-the-art efforts in tackling the CASH problem. In addition, we highlight the research work of automating the other steps of the full end-to-end machine learning pipeline (AutoML) from data understanding (pre-modeling) till model deployment (post-modeling)[5]. The remainder of this paper is organized as follows. Section 2 covers the various techniques that have been introduced to tackle the challenge of warm starting (meta-learning) for AutoML search problem in the context of machine learning and deep learning domains. Section 3 covers the various approaches that have been introduced for tackling the challenge of neural architecture search (NAS) in the context of deep learning. Section 4 focuses on the different approaches for automated hyper-parameter optimization. Section 5 comprehensively covers the various tools and frameworks that have been implemented to tackle the CASH problem. Section 6 covers the state-of-the-art research efforts on tackling the automation aspects for the other building blocks (Pre-modeling and Post-Modeling) of the complex machine learning pipeline. We discuss some of the research directions and open challenges that need to be addressed in order to achieve the vision and goals of the AutoML process in Section 7 before we finally conclude the paper in Section 8

## 2. META-LEARNING

In general, meta-learning can be described as the process of learning from previous experience gained during applying various learning algorithms on different kinds of data,

[5]We have prepared a repository with the state-of-the-art resources in the AutoML domain and made it available on `https://github.com/DataSystemsGroupUT/AutoML_Survey`

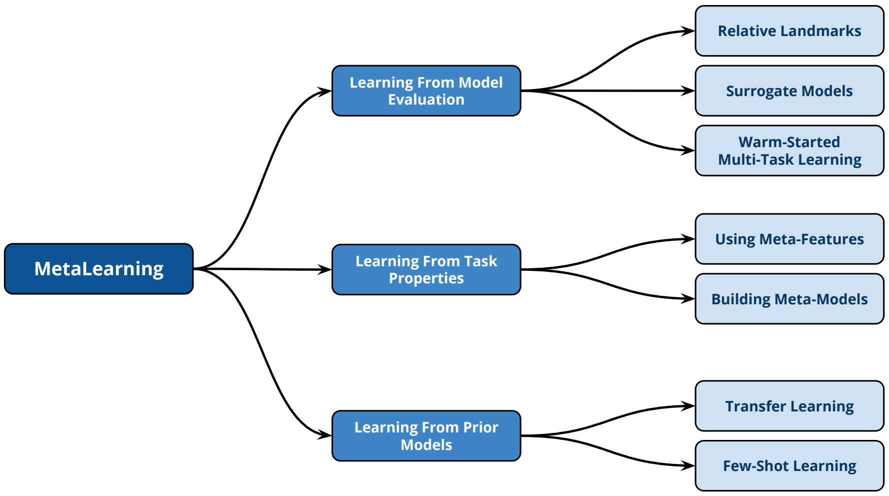

**Figure 3: A Taxonomy of Meta-Learning Techniques.**

and hence reducing the needed time to learn new tasks [18]. In the context of machine learning, several *meta learning*-techniques have been introduced as an effective mechanism to tackle the challenge of warm start for optimization algorithms. These techniques can generally be categorized into three broad groups [123]: *learning based on task properties*, *learning from previous model evaluations* and *learning from already pretrained models* (Figure 3).

One group of meta-learning techniques has been based on learning from task properties using the *meta-features* that characterize a particular dataset [78]. Generally speaking, each prior task is characterized by a feature vector, of $k$ features, $m(t_j)$. Simply, information from a prior task $t_j$ can be transferred to a new task $t_{new}$ based on their similarity, where this similarity between $t_{new}$ and $t_j$ can be calculated based on the distance between their corresponding feature vectors. In addition, a meta learner $L$ can be trained on the feature vectors of prior tasks along with their evaluations $\mathbf{P}$ to predict the performance of configurations $\theta_i$ on $t_{new}$.

Some of the commonly used meta features for describing datasets are simple meta features including number of instances, number of features, statistical features (e.g., skewness, kurtosis, correlation, co-variance, minimum, maximum, average), landmark features (e.g., performance of some landmark learning algorithms on a sample of the dataset), and information theoretic features (e.g., the entropy of class labels) [123]. In practice, the selection of the best set of meta features to be used is highly dependent on the application [15]. When computing the similarity between two tasks represented as two feature vectors of meta data, it is important to normalize these vectors or apply dimensionality reduction techniques such as principle component analysis [15, 5]. Another way to extract meta-features is to learn a joint distribution representation for set of tasks. One way to learn meta features is as follows:

For a particular set of configurations $\theta_i$, evaluate all prior tasks $t_j$ and for each pair of configurations $\theta_a$ and $\theta_b$, generate a binary meta feature $m_{j,a,b}$ which is equal to one if the configuration $\theta_a$ outperforms the configuration $\theta_b$. The meta feature $m_{new,a,b}$ for a particular new task $t_{new}$ can be computed by learning meta-rules from every pair of configurations $\theta_a$ and $\theta_b$. Each of these learnt rules predicts whether a particular configuration $\theta_a$ outperforms configuration $\theta_b$ on prior tasks $t_j$, given their meta features.

Another way to learn from prior tasks properties is through building *meta-models*. In this process, the aim is to build a meta model $L$ that learns complex relationships between meta features of prior tasks $t_j$. For a new task $t_{new}$, given the meta features for task $t_{new}$, model $L$ is used to recommend the best configurations. There exists a rich literature on using meta models for model configuration recommendations [114, 92, 65, 68, 44]. Meta models can also be used to rank a particular set of configurations by using the $K-$nearest neighbour model on the meta features of prior tasks and predicting the top $k$ tasks that are similar to new task $t_{new}$ and then ranking the best set of configurations of these similar tasks [19, 32]. Moreover, they can also be used to predict the performance of new task based on a particular configuration [105, 45]. This gives an indication about how good or bad this configuration can be, and whether it is worth evaluating it on a particular new task.

Another group of meta-learning techniques are based on *learning from previous model evaluation*. In this context, the problem is formally defined as follows.

Given a set of machine learning tasks $t_j \in T$, their corresponding learned models along their hyper-parameters $\theta \in \Theta$ and $P_{i,j} = P(\theta_i, t_j)$, the problem is to learn a meta-learner $L$ that is trained on meta-data $\mathbf{P} \cup \mathbf{P}_{new}$ to predict recommended configuration $\Theta^*_{new}$ for a new task $t_{new}$, where $T$ is the set of all prior machine learning tasks. $\Theta$ is the configuration space (hyper-parameter setting, pipeline components, network architecture, and network hyper-parameter), $\Theta_{new}$ is the configuration space for a new machine learning task $t_{new}$, $\mathbf{P}$ is the set of all prior evaluations $P_{i,j}$ of configuration $\theta_i$ on a prior task $t_j$, and $\mathbf{P}_{new}$ is a set of evaluations $P_{i,new}$ for a new task $t_{new}$.

One way to get hyper-parameter recommendations for a new task $t_{new}$ is to recommend based on similar prior tasks $t_j$. In the following, we will go through three different ways: *relative landmarks*, *surrogate models* and *Warm-Started Multi-task Learning* for relating similarity between $t_{new}$ and $t_j$. One way to measure the similarity between $t_{new}$ and $t_j$ is using the *relative landmarks* that measures the performance difference between two model configurations on the same task [42]. Two tasks $t_{new}$ and $t_j$ are considered similar if their relative landmarks performance of the considered configurations are also similar. Once similar tasks have been identified, a meta learner can be trained on the evaluations $P_{i,j}$ and $P_{i,new}$ to recommend new configurations for task $t_{new}$. Another way to define learning from model evaluations is through *surrogate models* [129]. In particular, *Surrogate models* get trained on all prior evaluations of for all prior tasks $t_j$. Simply, for a particular task $t_j$, if the surrogate model can predict accurate configuration for a new task $t_new$, then tasks $t_new$ and $t_j$ are considered similar. *Warm-Started Multi-task Learning* is another way to capture similarity between $t_j$ and $t_{new}$. Warm-Started Multi-task Learning uses the set of prior evaluations $\mathbf{P}$ to learn a joint task representation [97] which is used to train surrogate models on prior tasks $t_j$ and integrate them to a feed-forward neural network to learn a joint task representation that can predict accurately a set of evaluation $t_{new}$.

Learning from prior models can be done using *Transfer learning* [94], which is the process of utilization of pretrained models on prior tasks $t_j$ to be adapted on a new task $t_{new}$, where tasks $t_j$ and $t_{new}$ are similar. Transfer learning has received lots of attention especially in the area of neural network. In particular, neural network architecture and neural network parameters are trained on prior task $t_j$ that can be used as an initialization for model adaptation on a new task $t_{new}$. Then, the model can be fine-tuned [8, 6, 22]. It has been shown that neural networks trained on big image datasets such as ImageNet [68] can be transferred as well to new tasks [108, 31]. Transfer learning usually works well when the new task to be learned is similar to the prior tasks, otherwise transfer learning may lead to unsatisfactory results [131]. In addition, prior models can be used in *Few-Shot Learning* where a model is required to be trained using a few training instances given the prior experience gained from already trained models on similar tasks. For instance, learning a common feature representation for different tasks can be used to take the advantage of utilizing pretrained models that can initially guide the model parameters optimization for the few instances available. Some attempts have tackled the few-shot learning. For example, Snell et al. [111] presented an approach where prototypical networks were used to map instances from separate tasks into a similar dimensional space to use the same model. Moreover, Ravi and Larochelle [101] proposed an LSTM network as a meta-learner that is trained to learn the update rule for fitting the neural network learner. For instance, the loss and gradient are passed from the learner to the meta-learner networks which in turn updates them before modifying the learner parameters. Mishra et al. [87] proposed a meta-learner that tries to learn a common feature vector among different tasks. The architecture of this meta-learner consists of an architecture of convolution layers in addition to attention layers that tries to learn useful parts from tasks that can be used to make it more generic for new tasks.

## 3. NEURAL ARCHITECTURE SEARCH FOR DEEP LEARNING

In general, deep learning techniques represent a subset of machine learning methodologies that are based on artificial neural networks (ANN) which are mainly inspired by the neuron structure of the human brain [9]. It is described as *deep* because it has more than one layer of nonlinear feature transformation. Neural Architecture Search (NAS) is a fundamental step in automating the machine learning process and has been successfully used to design the model architecture for image and language tasks [136, 137, 21, 73, 74]. Broadly, NAS techniques falls into five main categories including *random search*, *reinforcement learning*, *gradient-based methods*, *evolutionary methods*, and *Bayesian optimization* (Figure 4).

*Random search* is one of the most naive and simplest approaches for network architecture search. For example, Hoffer el al. [48] have presented an approach to find good network architecture using random search combined with well-trained set of shared weights. Li et Talwalkar [70] proposed new network architecture search baselines that are based on random search with early-stopping for hyper-parameter optimization. Results show that random search along with early-stopping achieves the state-of-the-art network architecture search results on two standard NAS bookmarkers which are PTB and CIFAR-10 datasets.

*Reinforcement learning* [118] is another approach that has been used to find the best network architecture. Zoph and Le [136] used a recurrent neural network (LSTM) with reinforcement to compose neural network architecture. More specifically, recurrent neural network is trained through a gradient based search algorithm called `REINFORCE` [128] to maximize the expected accuracy of the generated neural network architecture. Baker et al. [4] introduced a meta-modeling algorithm called `MetaQNN` based on reinforcement learning to automatically generate the architecture of convolutional neural network for a new task. The convolutional neural network layers are chosen sequentially by a learning agent that is trained using $Q-$learning with $\epsilon-$greedy exploration technique. Simply, the agent explores a finite search space of a set of architectures and iteratively figures out architecture designs with improved performance on the new task to be learnt.

*Gradient-based optimization* is another common way for neural network architecture search. Liu et al. [75] proposed an approach based on continuous relaxation of the neural architecture allowing using a gradient descent for architecture search. Experiments showed that this approach excels in finding high-performance convolutional architectures for image classification tasks on CIFAR-10, and ImageNet datasets. Shin et al. [109] proposed a gradient-based optimization approach for learning the network architecture and parameters simultaneously. Ahmed and Torresani [1] used gradient based approach to learn network architecture. Experimental results on two different networks architecture ResNet and ResNeXt show that this approach yields to better accuracy and significant reduction in the number of parameters.

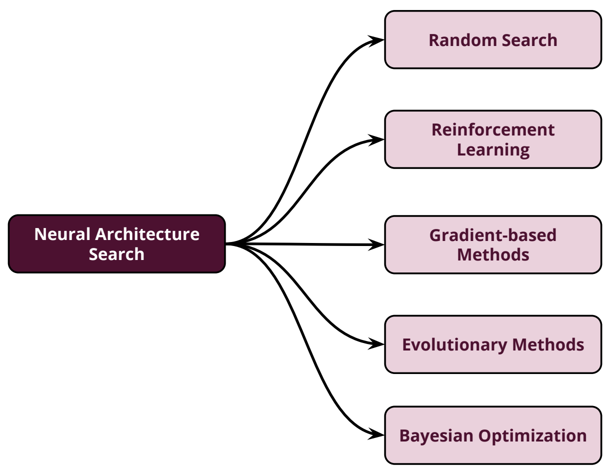

**Figure 4: A Taxonomy for the Neural Network Architecture Search (NAS) Techniques**

Another direction for architecture search is *evolutionary algorithms* which are well suited for optimizing arbitrary structure. Miller et al. [86] considered an evolutionary algorithm to propose the architecture of the neural network and network weights as well. Many evolutionary approaches based on genetic algorithms are used to optimize the neural networks architecture and weights [117, 116, 2] while others rely on hierarchical evolution [74]. Some recent approaches consider using the multi-objective evolutionary architecture search to optimize training time, complexity and performance [76, 36] of the network. `LEAF` [72] is an evolutionary AutoML framework that optimizes hyper-parameters, network architecture and the size of the network. LEAF uses `CoDeepNEAT` [85] which is a powerful evolutionary algorithm based on `NEAT` [104]. LEAF achieved the state-of-the-art performance results on medical image classification and natural language analysis. For supervised learning tasks, evolutionary based approaches tend to outperform reinforcement learning approaches especially when the neural network architecture is very complex due to having millions of parameters to be tuned. For example, the best performance achieved on ImageNet and CIFAR-10 has been obtained using evolutionary techniques [103].

*Bayesian optimization* based on Gaussian processes has been used by Kandasamy et al. [57] and Swersky et al. [120] for tackling the neural architecture search problem. In addition, lots of work focused on using tree based models such as random forests and tree Parzen estimators [13] to effectively optimize the network architecture as well as its hyper-parameters [12, 30, 82]. Bayesian optimization may outperform evolutionary algorithms in some problems as well [62].

## 4. HYPER-PARAMETER OPTIMIZATION

After choosing the model pipeline algorithm(s) with the highest potential for achieving the top performance on the input dataset, the next step is tuning the hyper-parameters of the model in order to further optimize the model performance. It is worth mentioning that some tools have democratized the space of different learning algorithms in discrete number of model pipelines [66, 39]. So, the model selection itself can be considered as a categorical parameter that needs to be tuned in the first place before modifying its hyper-parameters. In general, several hyper-parameter optimization techniques have been based and borrowed ideas from the domains of statistical model selection and traditional optimization techniques [25, 96, 99]. In principle, the automated hyper-parameter tuning techniques can be classified into two main categories: *black-box optimization techniques* and *multi-fidelity optimization techniques* (Figure 5).

### 4.1 Black-Box optimization

*Grid search* is a simple basic solution for the hyper-parameter optimization [90] in which all combinations of hyper-parameters are evaluated. Thus, grid search is computationally expensive, infeasible and suffers from the *curse of dimensionality* as the number of trails grows exponentially with the number of hyper-parameters [7]. Another alternative is *random search* in which it samples configurations at random until a particular budget $B$ is exhausted [10]. Given a particular computational budget $B$, random search tends to find better solutions than grid search [90]. One of the main advantages of random search, and grid search is that they can be easily parallelized over a number of workers which is essential when dealing with big data.

*Bayesian Optimization* is one of the state-of-the-art black-box optimization techniques which is tailored for expensive objective functions [69, 134, 88, 55]. Bayesian optimization has received huge attention from the machine learning community in tuning deep neural networks for different tasks including classification tasks [113, 112], speech recognition [24] and natural language processing [81]. Bayesian optimization consists of two main components which are surrogate models for modeling the objective function and an acquisition function that measures the value that would be generated by the evaluation of the objective function at a new point. Gaussian processes have become the standard surrogate for modeling the objective function in Bayesian optimization [112, 80]. One of the main limitations of the Gaussian processes is the cubic complexity to the number of data points which limits their parallelization capability. Another limitation is the poor scalability when using the standard kernels. Random forests [20] are another choice for modeling the objective function in Bayesian optimization. First, the algorithm starts with growing $B$ regression trees, each of which is built using $n$ randomly selected data points with replacement from training data of size $n$. For each tree, a split node is chosen from $d$ algorithm parameters. The minimum number of points are considered for further split are set to 10 and the number of trees $B$ to grow is set be 10 to maintain low computational overhead. Then, the random forest predicted mean and variance for each new configuration is computed. The random forests' complexity of the fitting and predicting variances are $O(n \log n)$ and $O(\log n)$ respectively which is much better compared to the Gaussian process. Random forests are used by the Sequential Model-based Algorithm Configuration (SMAC) library [51]. In general Tree-structured Parzen Estimator (TPE) [13] does not define a predictive distribution over the objective function but it creates two density functions that act as generative models for all domain variables. Given a percentile $\alpha$, the observations are partitioned into two sets of observations (good observations and bad ob-

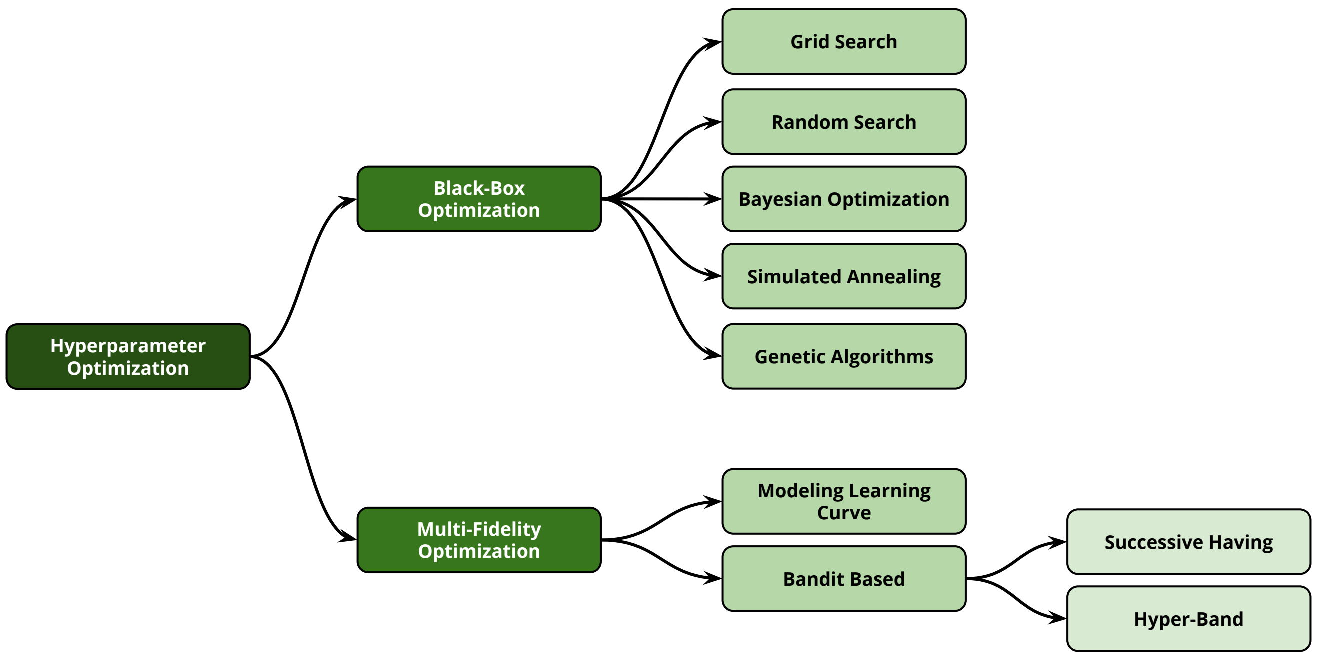

**Figure 5: A Taxonomy for the Hyper-parameter Optimization Techniques.**

servations) where simple Parzen windows are used to model the two sets. The ratio between the two density functions reflects the expected improvement in the acquisition function and is used to recommend new configurations for hyper-parameters. Tree-Structured Parzen estimator (TPE) has shown great performance for hyper-parameter optimization tasks [13, 12, 35, 37, 115].

*Simulated Annealing* is a hyper-parameter optimization approach which is inspired by the metallurgy technique of heating and controlled cooling of materials [61]. This optimization techniques goes through a number of steps. First, it randomly chooses a single value (current state) to be applied to all hyper-parameters and then evaluates model performance based on it. Second, it randomly updates the value of one of the hyper-parameters by picking a value from the immediate neighborhood to get neighboring state. Third, it evaluates the model performance based on the neighboring state. Forth, it compares the performance obtained from the current and neighbouring states. Then, the user chooses to reject or accept the neighbouring state as a current state based on some criteria.

*Genetic Algorithms* (GA) are inspired by the process of natural selection [49]. The main idea of genetic-based optimization techniques is simply applying multiple genetic operations to a population of configurations. For example, the *crossover* operation simply takes two parent *chromosomes* (configurations) and combines their genetic information to generate new *offspring*. More specifically, the two parents configurations are cut at the same crossover point. Then, the sub-parts to the right of that point are swapped between the two parents chromosomes. This contributes to two new *offspring* (child configuration). Mutation randomly chooses a chromosome and mutates one or more of its parameters that results in totally new chromosome.

## 4.2 Multi-fidelity optimization

Multi-fidelity optimization is an optimization technique which focuses on decreasing the evaluation cost by combining a large number of cheap low-fidelity evaluations and a small number of expensive high-fidelity evaluation [38, 79, 50]. In practice, such optimization technique is essential when dealing with big datasets as training one hyper-parameter may take days. More specifically, in multi-fidelity optimization, we can evaluate samples in different levels. For example, we may have two evaluation functions: *high-fidelity* evaluation and *low-fidelity* evaluation. The high-fidelity evaluation outputs precise evaluation from the whole dataset. On the other hand, the low-fidelity evaluation is a cheaper evaluation from a subset of the dataset. The idea behind the multi-fidelity evaluation is to use many low-fidelity evaluation to reduce the total evaluation cost. Although the low fidelity optimization results in cheaper evaluation cost that may suffer from optimization performance, but the speedup achieved is more significant than the approximation error.

*Modeling learning curves* is an optimization technique that models learning curves during hyper-parameter optimization and decides whether to allocate more resources or to stop the training procedure for a particular configuration. For example, a curve may model the performance of a particular hyper-parameter on an increasing subset of the dataset. Learning curve extrapolation is used in predicting early termination for a particular configuration [55]; the learning process is terminated if the performance of the predicted configuration is less than the performance of the best model trained so far in the optimization process. Combining early predictive termination criterion with Bayesian optimization leads to more reduction in the model error rate than the vanilla Bayesian black-box optimization. In addition, such technique resulted in speeding-up the optimization by a factor of 2 and achieved the state-of-the-art neural network on CIFAR-10 dataset [30].

*Bandit-based algorithms* have shown to be powerful in tackling deep learning optimization challenges. In the following, we consider two strategies of the bandit-based techniques which are the *Successive halving* and *HyperBand*. *Successive halving* is a bandit-based powerful multi-fidelity technique in which given a budget $B$, first, all the configurations are evaluated. Next, they are ranked based on their performance. Then, half of these configurations that performed worse than the others are removed. Finally, the budget of the previous steps are doubled and repeated until only one algorithm remains. It is shown that the successive halving outperforms the uniform budget allocation technique in terms of the computation time, and the number of iterations required [52]. On the other hand, successive halving suffer from the following problem. Given a time budget $B$, the user has to choose, in advance, whether to consume the larger portion of the budget exploring a large number of configurations while spending a small portion of the time budget on tuning each of them or to consume the large portion of the budget on exploring few configurations while spending the larger portion of the budget on tuning them.

*HyperBand* is another bandit-based powerful multi-fidelity hedging technique that optimizes the search space when selecting from randomly sampled configurations [71]. More specifically, partition a given budget $B$ into combinations of number of configurations and budget assigned to each configuration. Then, call successive halving technique on each random sample configuration. Hyper-Band shows great success with deep neural networks and perform better than random search and Bayesian optimization.

## 5. TOOLS AND FRAMEWORKS

In this section, we provide a comprehensive overview of several tools and frameworks that have been implemented to automate the process of combined algorithm selection and hyper-parameter optimization process. In general, these tools and frameworks can be classified into three main categories: *centralized*, *distributed*, and *cloud-based*.

### 5.1 Centralized Frameworks

Several tools have been implemented on top of widely used *centralized* machine learning packages which are designed to run in a *single* node (machine). In general, these tools are suitable for handling small and medium sized datasets. For example, *Auto-Weka*[6] is considered as the first and pioneer machine learning automation framework [66]. It was implemented in Java on top of `Weka`[7], a popular machine learning library that has a wide range of machine learning algorithms. `Auto-Weka` applies Bayesian optimization using Sequential Model-based Algorithm Configuration (`SMAC`) [51] and tree-structured parzen estimator (TPE) for both algorithm selection and hyper-parameter optimization (Auto-Weka uses SMAC as its default optimization algorithm but the user can configure the tool to use TPE). In particular, SMAC tries to draw the relation between algorithm performance and a given set of hyper-parameters by estimating the predictive mean and variance of their performance along the trees of a random forest model. The main advantage of using SMAC is its robustness by having the ability to discard low performance parameter configurations quickly after the evaluation on low number of dataset folds. SMAC shows better performance on experimental results compared to TPE [51].

$Auto-MEKA_{GGP}$ [26] focuses on the AutoML task for multi-label classification problem [121] that aims to learn models from data capable of representing the relationships between input attributes and a set of class labels, where each instance may belong to more than one class. Multi-label classification has lots of applications especially in medical diagnosis in which a patient may be diagnosed with more than one disease. $Auto-MEKA_{GGP}$ is a grammar-based genetic programming framework that can handle complex multi-label classification search space and simply explores the hierarchical structure of the problem. $Auto-MEKA_{GGP}$ takes as input both of the dataset and a grammar describing the hierarchical search space of the hyper-parameters and the learning algorithms from `MEKA`[8] framework [102]. $Auto-MEKA_{GGP}$ starts by creating an initial set of trees representing the multi-label classification algorithms by randomly choosing valid rules from the grammar, followed by the generation of derivation trees. Next, map each derivation tree to a specific multi-label classification algorithm. The initial trees are evaluated on the input dataset by running the learning algorithm, they represent, using MEKA framework. The quality of the individuals are assessed using different measures such as fitness function. If a stopping condition is satisfied (e.g. a quality criteria ), a set of individuals (trees) are selected in a tournament selection. Crossover and mutation are applied in a way that respect the grammar constraints on the selected individuals to create a new population. At the end of the evolution, the best set of individuals representing the well performing set of multi-label tuned classifiers are returned.

*Auto-Sklearn*[9] has been implemented on top of `Scikit-Learn`[10], a popular Python machine learning package [39]. `Auto-Sklearn` introduced the idea of meta-learning in the initialization of combined algorithm selection and hyper-parameter tuning. It used SMAC as a Bayesian optimization technique too. In addition, ensemble methods were used to improve the performance of output models. Both meta-learning and ensemble methods improved the performance of *vanilla* SMAC optimization. *hyperopt-Sklearn* [64] is another AutoML framework which is based on Scikit-learn machine learning library. Hyperopt-Sklearn uses `Hyperopt` [11] to define the search space over the possible Scikit-Learn main components including the learning and preprocessing algorithms. Hyperpot supports different optimization techniques including random search, and different Bayesian optimizations for exploring the search spaces which are characterized by different types of variables including categorical, ordinal and continuous.

[6]`https://www.cs.ubc.ca/labs/beta/Projects/autoweka/`
[7]`https://www.cs.waikato.ac.nz/ml/weka/`

[8]`http://waikato.github.io/meka/`
[9]`https://github.com/automl/auto-sklearn`
[10]`https://scikit-learn.org/`

*TPOT*[11] framework represents another type of solutions that has been implemented on top of Scikit-Learn [93]. It is based on genetic programming by exploring many different possible pipelines of feature engineering and learning algorithms. Then, it finds the best one out of them. *Recipe* [28] follows the same optimization procedure as TPOT using genetic programming, which in turn exploits the advantages of a global search. However, it considers the unconstrained search problem in TPOT, where resources can be spent into generating and evaluating invalid solutions by adding a grammar that avoids the generation of invalid pipelines, and can speed up optimization process. Second, it works with a bigger search space of different model configurations than *Auto-SkLearn* and TPOT.

*ML-Plan*[12] has been proposed to tackle the composability challenge on building machine learning pipelines [89]. In particular, it integrates a super-set of both Weka and Scikit-Learn algorithms to construct a full pipeline. ML-Plan tackles the challenge of the search problem for finding optimal machine learning pipeline using hierarchical task network algorithm where the search space is modeled as a large tree graph where each leaf node is considered as a goal node of a full pipeline. The graph traversal starts from the root node to one of the leaves by selecting some random paths. The quality of a certain node in this graph is measured after making $n$ such random complete traversals and taking the minimum as an estimate for the best possible solution that can be found. The initial results of this approach has shown that the composable pipelines over Weka and Scikit-Learn does not significantly outperform the outcomes from Auto-Weka and Auto-Sklearn frameworks because it has to deal with larger search space.

*SmartML*[13] has been introduced as the first R package for automated model building for classification tasks [78]. Figure 6 illustrate the framework architecture of SmartML. In the algorithm selection phase, SmartML uses a meta-learning approach where the meta-features of the input dataset is extracted and compared with the meta-features of the datasets that are stored in the framework's knowledge base, populated from the results of the previous runs. The similarity search process is used to identify the similar datasets in the knowledge base, using a nearest neighbor approach, where the retrieved results are used to identify the best performing algorithms on those similar datasets in order to nominate the candidate algorithms for the dataset at hand. The hyper-parameter tuning of SmartML is based on SMAC Bayesian Optimisation [51]. SmartML maintains the results of the new runs to continuously enrich its knowledge base with the aim of further improving the accuracy of the similarity search and thus the performance and robustness for future runs.

*Autostacker* [23] is an AutoML framework that uses an evolutionary algorithm with hierarchical stacking for efficient hyper-parameters search. Autostacker is able to find pipelines, consisting of preprocessing, feature engineering and machine learning algorithms with the best set of hyper-parameters, rather than finding a single machine learning model with the best set of hyper-parameters. Autostacker generates cascaded architectures that allow the components of a pipeline to "correct mistakes made by each other" and hence improves the overall performance of the pipeline. Autostacker simply starts by selecting a set of pipelines randomly. Those pipelines are fed into an evolutionary algorithm that generates the set of winning pipelines.

*AlphaD3M* [33] has been introduced as an AutoML framework that uses meta reinforcement learning to find the most promising pipelines. AlphaD3M finds patterns in the components of the pipelines using recurrent neural networks, specifically long short term memory (LSTM) and Monte-Carlo tree search in an iterative process which is computationally efficient in large search space. In particular, for a given machine learning task over a certain dataset, the network predicts the actions probabilities which lead to sequences that describe the whole pipeline. The predictions of the LSTM neural network are used by Monte-Carlo tree search by running multiple simulations to find the best pipeline sequence.

*OBOE*[14] is an AutoML framework for time constrained model selection and hyper-parameter tuning [130]. OBOE finds the most promising machine learning model along with the best set of hyper-parameters using collaborative filtering. OBOE starts by constructing an *error matrix* for some base set of machine learning algorithms, where each row represents a dataset and each column represents a machine learning algorithm. Each cell in the matrix represents the performance of a particular machine learning model along with its hyper-parameters on a specific dataset. In addition, OBOE keeps track of the running time of each model on a particular dataset and trains a model to predict the running time of a particular model based on the size and the features of the dataset. Simply, a new dataset is considered as a new row in the error matrix. In order to find the best machine learning algorithm for a new dataset, OBOE runs a particular set of models corresponding to a subset of columns in the error matrix which are predicted to run efficiently on the new dataset. In order to find the rest of the entries in the row, the performance of the models that have not been evaluated are predicted. The good thing about this approach is that it infers the performance of lots of models without the need to run them or even computing meta-features and that is why OBOE can find well performing model within a reasonable time budget.

The *PMF*[15] AutoML framework is based on collaborative filtering and Bayesian optimization [43]. More specifically, the problem of selecting the best performing pipeline for a specific task is modeled as a collaborative filtering problem that is solved using probabilistic matrix factorization techniques. PMF considers two datasets to be similar if they have similar evaluations on a few set of pipelines and hence it is more likely that these datasets will have similar evaluations on the rest of the pipelines. This concept is quite related to collaborative filtering for movie recommendation in which users that had the same preference in the past are

[11]https://automl.info/tpot/
[12]https://github.com/fmohr/ML-Plan
[13]https://github.com/DataSystemsGroupUT/SmartML
[14]https://github.com/udellgroup/oboe/tree/master/automl
[15]https://github.com/rsheth80/pmf-automl

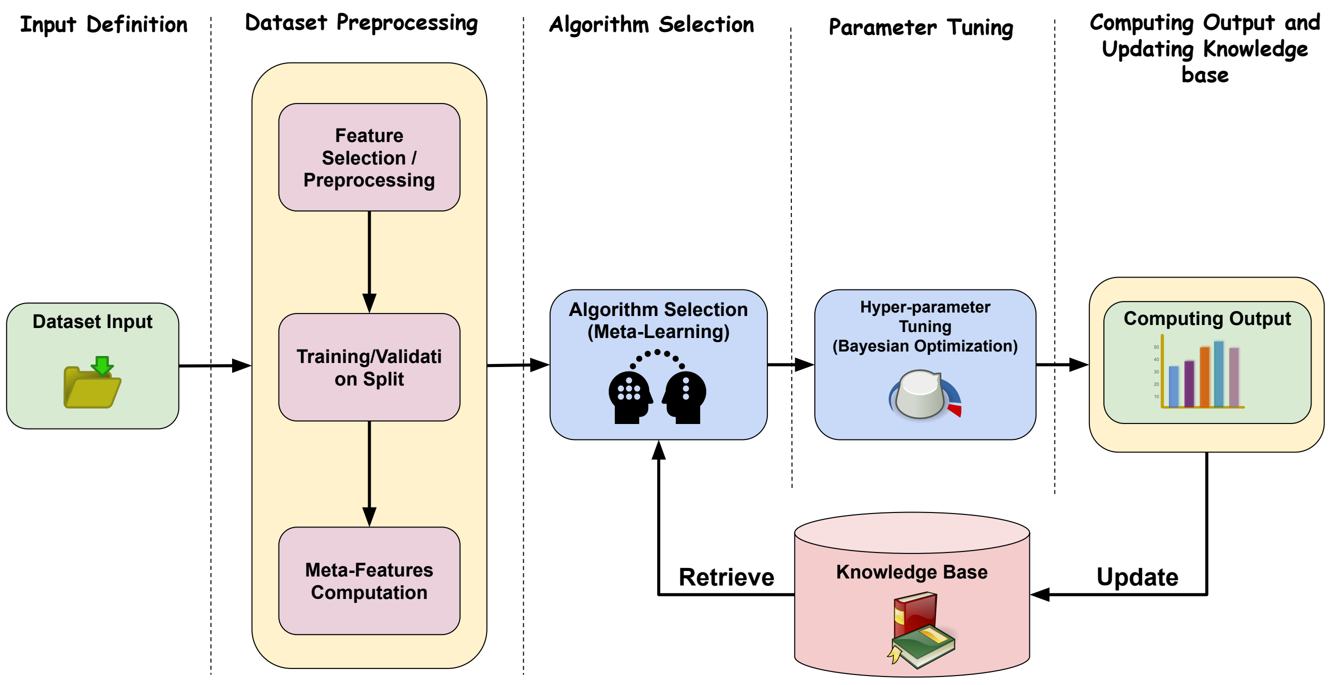
Input Definition
Dataset Preprocessing
Algorithm Selection
Parameter Tuning
Computing Output and Updating Knowledge base
Dataset Input
Feature Selection / Preprocessing
Training/Validation Split
Meta-Features Computation
Algorithm Selection (Meta-Learning)
Hyper-parameter Tuning (Bayesian Optimization)
Computing Output
Knowledge Base
Retrieve
Update

**Figure 6: SmartML: Framework Architecture.**

more likely to have the same preference in the future. In particular, the PMF framework trains each machine learning pipeline on a sample of each dataset and then evaluates such pipeline. This results in a matrix that summarizes the performance (accuracy or balanced accuracy for classification tasks and RMSE for regression tasks) of each machine learning pipeline of each dataset. The problem of predicting the performance of a particular pipeline on a new dataset can be mapped into a matrix factorization problem.

*VDS* [107] has been recently introduced as an *interactive* automated machine learning tool, that followed the ideas of a previous work on the `MLBase` framework [67]. In particular, it uses a meta learning mechanism (knowledge from the previous runs) to provide the user with a quick feedback, in few seconds, with an initial model recommendation that can achieve a reasonable accuracy while, on the back-end, conducting an optimization process so that it can recommend to the user more models with better accuracies, as it progresses with the search process over the search space. The VDS framework combines cost-based Multi-Armed Bandits and Bayesian optimizations for exploring the search space while using a rule-based search-space as query optimization technique. VDS prunes unpromising pipelines in early stages using an adaptive pipeline selection algorithm. In addition, it supports wide range of machine learning tasks including classification, regression, community detection, graph matching, image classification, and collaborative filtering. Table 1 shows a summary of the main features of the centralized state-of-the-art AutoML frameworks.

## 5.2 Distributed Frameworks

As the size of the dataset increases, solving the $CASH$ problem in a centralized manner turns out to be infeasible due to the limited computing resources (e.g, Memory, CPU) of a single machine. Thus, there is a clear need for distributed solutions that can harness the power of computing clusters that have multiple nodes to tackle the computational complexity of the problem. *MLbase*[16] has been the first work to introduce the idea of developing a distributed framework of machine learning algorithm selection and hyperparameter optimization [67]. MLbase has been based on `MLlib` [83], a Spark-based ML library. It attempted to reused cost-based query optimization techniques to prune the search space at the level of *logical learning plan* before transforming it into a *physical learning plan* to be executed.

Figure 7 illustrates the architectures of the *Auto-Tuned Models (ATM)* framework[17] that has been introduced as a parallel framework for fast optimization of machine learning modeling pipelines [119]. In particular, this framework depends on parallel execution along multiple nodes with a shared model hub that stores the results out of these executions and try to enhance the selection of other pipelines that can out perform the current chosen ones. The user can decide to use either of ATM's two searching methods, a hybrid Bayesian and multi-armed bandit optimization system, or a model recommendation system that works by exploiting the previous performance of modeling techniques on a variety of datasets.

*`TransmogrifAI`*[18] is one of the most recent modular tools written in Scala. It is built using workflows of feature preprocessors, and model selectors on top of Spark with mini-

[16]`http://www.mlbase.org/`
[17]`https://github.com/HDI-Project/ATM`
[18]`https://transmogrif.ai/`

Table 1: Summary of the Main Features of Centralized AutoML Frameworks

| | **Release Date** | **Core Language** | **Training Framework** | **Optimization Technique** | **ML Task** | **Meta Learning** | **User Interface** | **Automatic Feature Extraction** | **Open Source** |
|---|---|---|---|---|---|---|---|---|---|
| **AutoWeka** | 2013 | Java | Weka | Bayesian optimization | Single-label classification regression | × | ✓ | ✓ | ✓ |
| **AutoSklearn** | 2015 | Python | scikit-learn, | Bayesian optimization | Single-label classification regression | ✓ | × | ✓ | ✓ |
| **TPOT** | 2016 | Python | scikit-learn | Genetic Algorithm | Single-label classification regression | × | × | ✓ | ✓ |
| **SmartML** | 2019 | R | mlr, RWeka & other R packages | Bayesian optimization | Single-label classification | ✓ | ✓ | × | ✓ |
| **Auto-MEKA**$_{GGP}$ | 2018 | Java | Meka | Grammar-based genetic algorithm | Multi-label classification | ✓ | × | × | ✓ |
| **Recipe** | 2017 | Python | scikit-learn | Grammar-based genetic algorithm | Single-label classification | ✓ | × | ✓ | ✓ |
| **MLPlan** | 2018 | Java | Weka and scikit-learn | Hierachical Task Planning | Single-label classification | × | × | ✓ | ✓ |
| **Hyperopt-sklearn** | 2014 | Python | scikit-learn | Bayesian Optimization & Random Search | Single-label classification regression | × | × | ✓ | ✓ |
| **Autostacker** | 2018 | - | - | Genetic Algorithm | Single-label classification | × | × | ✓ | × |
| **VDS** | 2019 | - | - | cost-based Multi-Armed Bandits and Bayesian Optimization | Single-label classification regression image classification audio classification graph matching | ✓ | ✓ | ✓ | × |
| **AlphaD3M** | 2018 | - | - | Reinforcement learning | Single-label classification regression | ✓ | × | ✓ | × |
| **OBOE** | 2019 | Python | scikit-learn | collaborative filtering | Single-label classification | ✓ | × | × | ✓ |
| **PMF** | 2018 | Python | scikit-learn | collaborative filtering & Bayesian optimization | Single-label classification | ✓ | × | ✓ | ✓ |

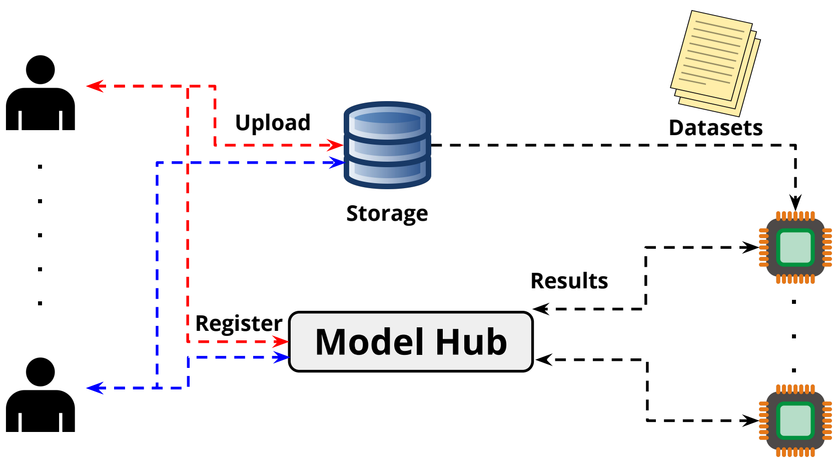

**Figure 7: ATM: Framework Architecture.**

mal human involvement. It has the ability to reuse the selected work-flows. Currently, `TransmogrifAI` supports eight different binary classifiers and five regression algorithms. *MLBox*[19] is a Python-based AutoML framework for distributed preprocessing, optimization and prediction. MLBox supports model stacking where a new model is trained from the combined predictors of multiple previously trained models. It uses `hyperopt`[20], a distributed asynchronous hyper-parameter optimization library, in Python, to perform the hyper-parameter optimisation process.

*`Rafiki`*[21] has been introduced as a distributed framework which is based on the idea of using previous models that achieved high performance on the same tasks [126]. In this framework, regarding the data and parameters storage, the data uploaded by user to be trained is stored in a Hadoop Distributed File System (HDFS). During training, there is a database for each model storing the best version of parameters from hyper-parameter tuning process. This database is kept in memory as it is accessed and updated frequently. Once the hyper-parameter tuning process is finished, the database is dumped to the disk. The types of parameters to be tuned are either related to model architecture like number of Layers, and Kernel or related to the training algorithm itself like weight decay, and learning rate. All these parameters can be tuned using random search or Bayesian optimization. Table 2 shows a summary of the main features of the distributed AutoML frameworks.

## 5.3 Cloud-Based Frameworks

Several cloud-based solutions have been introduced to tackle the automated machine learning problem using the availability of high computational power on cloud environments to try a wide range of models and configurations. *Google AutoML*[22] has been introduced as a block of the artificial intelligence platform services supported by Google cloud. It supports training a wide range of machine learning models in different domains with minimal user experience. These models can be trained for various tasks including sight, language, and structured data. For instance, AutoML vision, and video intelligence are used in getting insights from visual data like object localization, detection and classification for both static images, and video streams through already pretrained models or training custom models on user data. Similarly, AutoML Natural language, and AutoML translation provide user with APIs for automatic language detection, and transition in addition to insightful text analysis like sentiment classification, and entity extraction. These language services support ten different languages including English, Chinese, French, German and Russian. On the other hand, AutoML Tables supports training high quality models on tabular structured data by automating feature engineering, model selection, and hyper-parameter tuning steps. Both *Google AutoML* pretrained, and custom models are based on TensorFlow that mainly relies on Google's state-of-the-art transfer learning, neural architecture search technology, and Reinforcement learning with gradient policy upgrade.

*Azure AutoML*[23] is a cloud-based service that can be used to automate building machine learning pipeline for a both classification and regression tasks. AutoML Azure uses collaborative filtering and Bayesian optimization to search for the most promising pipelines efficiently [43] based on a database that is constructed by running millions of experiments of evaluation of different pipelines on many datasets. This database helps in finding the good solutions for new datasets quickly. *Azure AutoML* is available in the Python SDK of Microsoft Azure machine learning and it is based on scikit-learn search space of different learning algorithms. In addition, it gives the user the flexibility to use this service either locally or leveraging the performance and scalability of Azure cloud services.

*Amazon Sage Maker*[24] provides its users with a wide set of most popular machine learning, and deep learning frameworks to build their models in addition to automatic tuning

[19] `https://github.com/AxeldeRomblay/MLBox`
[20] `https://github.com/hyperopt/hyperopt`
[21] `https://github.com/nginyc/rafiki`
[22] `https://cloud.google.com/automl/`
[23] `https://docs.microsoft.com/en-us/azure/machine-learning/service/`
[24] `https://aws.amazon.com/machine-learning/`

Table 2: **Summary of the Main Features of Distributed AutoML Frameworks**

| | Release Date | Core Language | Optimization Technique | Training Framework | Meta-Learning | User Interface | Open Source |
|---|---|---|---|---|---|---|---|
| **MLBase** | 2013 | Scala | Cost-based Multi-Armed Bandits | Spark MLlib | × | × | × |
| **ATM** | 2017 | Python | Hybrid Bayesian, and Multi-armed bandits Optimization | Scikit-Learn | ✓ | × | ✓ |
| **MLBox** | 2017 | Python | Distributed Random search, Tree-Parzen estimators | Scikit-Learn Keras | × | × | ✓ |
| **Rafiki** | 2018 | Python | Distributed random search, Bayesian Optimization | TensorFlow Scikit-Learn | × | ✓ | ✓ |
| **TransmogrifAI** | 2018 | Scala | Bayesian Optimization, and Random Search | SparkML | × | × | ✓ |

for the model parameters. *Sage Maker* supports automatic deployment for models on auto-scaling clusters in multiple zones to ensure the high availability, and performance during generation of predictions. Moreover, Amazon offers a long list of pretrained models for different AI services that can be easily integrated to user applications including different image and video analysis, voice recognition, text analytics, forecasting, and recommendation systems.

## 5.4 Neural Network Automation Frameworks

Recently, some frameworks (e.g., `Auto-Keras` [53], and `Auto-Net` [82]) have been proposed with the aim of automatically finding neural network architectures that are competitive with architectures designed by human experts. However, the results so far are not significant. For example, *Auto-Keras* [53] is an open source efficient neural architecture search framework based on Bayesian optimization to guide the network morphism. In order to explore the search space efficiently, Auto-Keras uses a neural network kernel and tree structured acquisition function with iterative Bayesian optimization. First, a Gaussian process model is trained on the currently existing network architectures and their performance is recorded. Then, the next neural network architecture obtained by the acquisition function is generated and evaluated. Moreover, Auto-Keras runs in a parallel mode on both CPU and GPU.

*Auto-Net* [82] is an efficient neural architecture search framework based on SMAC optimization and built on top of `PyTorch`. The first version of Auto-Net is implemented within the Auto-sklearn in order to leverage some of the existing components of the of the machine learning pipeline in Auto-sklearn such as preprocessing. The first version of Auto Net only considers fully-connected feed-forward neural networks as they are applied on a large number of different datasets. Auto-net accesses deep learning techniques from Lasagne Python deep learning library [29]. Auto Net includes a number of algorithms for tuning the neural network weights including vanilla stochastic gradient descent , stochastic gradient descent with momentum, Adadelta [132], Adam [60], Nesterov momentum [91]and Adagrad [34].

*Neural Network Intelligence*(NNI)[25] is an open source toolkit by Microsoft that is used for tuning neural networks architecture and hyper-parameters in different environments including local machine, cloud and remote servers. NNI accelerates and simplifies the huge search space using built-in super-parameter selection algorithms including random search, naive evolutionary algorithms, simulated annealing, network morphism, grid search, hyper-band, and a bunch of Bayesian optimizations like SMAC [51], and BOHB [37]. NNI supports a large number of deep leaning frameworks including `PyTorch`, `TensorFlow`, `Keras`, `Caffe2`, `CNTK`, `Chainer` and `Theano`.

*DEvol* [26] is an open source framework for neural network architecture search that is based on genetic programming to evolve the number of layers, kernels, and filters, the activation function and dropout rate. DEvol uses parallel training in which multiple members of the population are evaluated across multiple GPU machines in order to accelerate the process of finding the most promising network.

*enas* [98] has been introduced as an open source framework for neural architecture search in Tensorflow based on reinforcement learning [136] where a controller of a recurrent neural network architecture is trained to search for optimal subgraphs from large computational graphs using policy gradient. Moreover, *enas* showed a large speed up in terms of GPU hours thanks to the sharing of parameters across child subgraphs during the search process.

*NAO* [77], and *Darts* [75] are open source frameworks for neural architecture search which propose a new continuous optimization algorithm that deals with the network architecture as a continuous space instead of the discretization followed by other approaches. In *NAO*, the search process starts by encoding an initial architecture to a continuous space. Then, a performance predictor based on gradient based optimization searches for a better architecture that is decoded at the end by a complementary algorithm to the encoder in order to map the continuous space found back into its architecture. On the other hand, *DARTS* learns new architectures with complex graph topologies from the rich continuous search space using a novel bilevel optimization algorithm. In addition, it can be applied to any specific architecture family without restrictions to any of convolutional and recurrent networks only. Both frameworks showed a competitive performance using limited computational resources compared with other neural architecture search frameworks.

*Evolutionary Neural AutoML for Deep Learning (LEAF)* [72]

[25]`https://github.com/Microsoft/nni`

[26]`https://github.com/joeddav/devol`

**Table 3: Summary of the Main Features of the n]Neural Architecture Search frameworks**

| | Release Date | Open Source | Optimization technique | Supported Frameworks | Interface |
|---|---|---|---|---|---|
| **Auto Keras** | 2018 | ✓ | Network Morphism | Keras | ✓ |
| **Auto Net** | 2016 | ✓ | SMAC | PyTorch | × |
| **NNI** | 2019 | ✓ | Random Search<br>Different Bayesian Optimizations<br>Annealing<br>Network Morphism<br>Hyper-Band<br>Naive Evolution<br>Grid Search | PyTorch,<br>TensorFlow,<br>Keras,<br>Caffe2,<br>CNTK,<br>Chainer<br>Theano | ✓ |
| **enas** | 2018 | ✓ | Reinforcement Learning | Tensorflow | × |
| **NAO** | 2018 | ✓ | Gradient based optimization | Tensorflow, PyTorch | × |
| **DARTS** | 2019 | ✓ | Gradient based optimization | PyTorch | × |
| **LEAF** | 2019 | × | Evolutionary Algorithms | - | × |

is an AutoML framework that optimizes neural network architecture and hyper-parameters using the state-of-the-art evolutionary algorithm and distributed computing framework. LEAF uses `CoDeepNEAT` [85] for optimizing deep neural network architecture and hyper-parameters. LEAF consists of three main layers which are algorithm layers, system layer and problem-domain layer. LEAF evolves deep neural networks architecture and hyper-parameters in the algorithm layer. The system layer is responsible for training the deep neural networks in a parallel mode on a cloud environment such as Microsoft Azure[27], Google Cloud[28] and Amazon AWS[29], which is essential in the evaluation of the fitness of the neural networks evolved in the algorithm layer. More specifically, the algorithm layer sends the neural network architecture to the system layer. Then, the system layer sends the evaluation of the fineness of this network back to the algorithm layer. Both the algorithm layer and the system layer work together to support the problem-domain layers where the problems of hyper-parameter tuning of network architecture search are solved. Table 3 shows summary of the main features of the state-of-the-art neural architecture search frameworks.

## 6. OTHER AUTOMATION ASPECTS IN MODEL BUILDING LIFE CYCLE

While current different AutoML tools and frameworks have minimized the role of data scientist in the modeling part and saved much effort, there are still several aspects that need human intervention and interpretability in order to make the correct decisions that can enhance and affect the modeling steps. These aspects belongs to two main building blocks of the machine learning production pipeline: *Pre-Modeling* and *Post-Modeling* (Figure 8). In general, *Pre-Modeling* is an important block of the machine learning pipeline that can dramatically affect the outcomes of the automated algorithm selection and hyper-parameters optimization process. The pre-modeling step includes a number of steps including data understanding, data preparation and data validation. In addition, the *Post-Modeling* block covers other important aspects including the management and deployment of produced machine learning model which represents a corner stone in the pipeline that requires the ability of packaging model for reproducibility. The aspects of these two building blocks can help on covering what is missed in current AutoML tools, and help data scientists in doing their job in a much easier, organized, and informative way. In this section, we give an overview of a number of systems and frameworks that have been developed to aid the data scientists on the steps of the pre-modeling and post-modeling blocks (Figure 9).

[27]https://azure.microsoft.com/en-us/
[28]https://cloud.google.com/
[29]https://aws.amazon.com/

## 6.1 Pre-Modeling

### *6.1.1 Data Understanding*

The data understanding step mainly focuses on formulating alerts that can be easily solved by identifying suitable actions. The key point of data understanding is the sensitivity of alerts to data errors. Available tools try to automate the process of data understanding and reduce human involvement in this stage. However, it is still essential to have a human to confirm the actions taken. There are three possible scenarios for data understanding:

**Sanity Checking**: The sanity checking process is used to ensure that the feature is suitable for being used to train a model. For example, a feature that has 70% missing values would not be a good one for training the model or the negative age values should be corrected. `seeDB` [125] has been introduced as a visualization recommendation engine that eases the analysis of data subsets in an efficient way by showing a large number of recommended visualizations in addition to choosing the appropriate metrics to measure the visualization interest in a near interactive performance with a reduced latency by over 100X than traditional techniques. `seeDB` can be run on top of a wide range of database management systems. `zenVisage` [110] extends the prior system `seeDB`. In addition to the visualization recommendation, it supports a graph based visual query language called ZQL which is a flexible technique to choose the desired insights from visualizations using very small lines. Moreover, `zenVisage` provides the user with an interactive interface which enables him to directly draw different chart types or specify the trend of current interest to be visualized. `QUDE` [133] presents a solution using a novel $\alpha$-investing technique to the multiple hypothesis testing error introduced by previous tools which is making several false discoveries based on inference seen by users from simple interactions visualized by these tools. `QUDE`has showed a significant decrease in the false discovery rate than other interactive tools for both synthetic and real-world datasets.

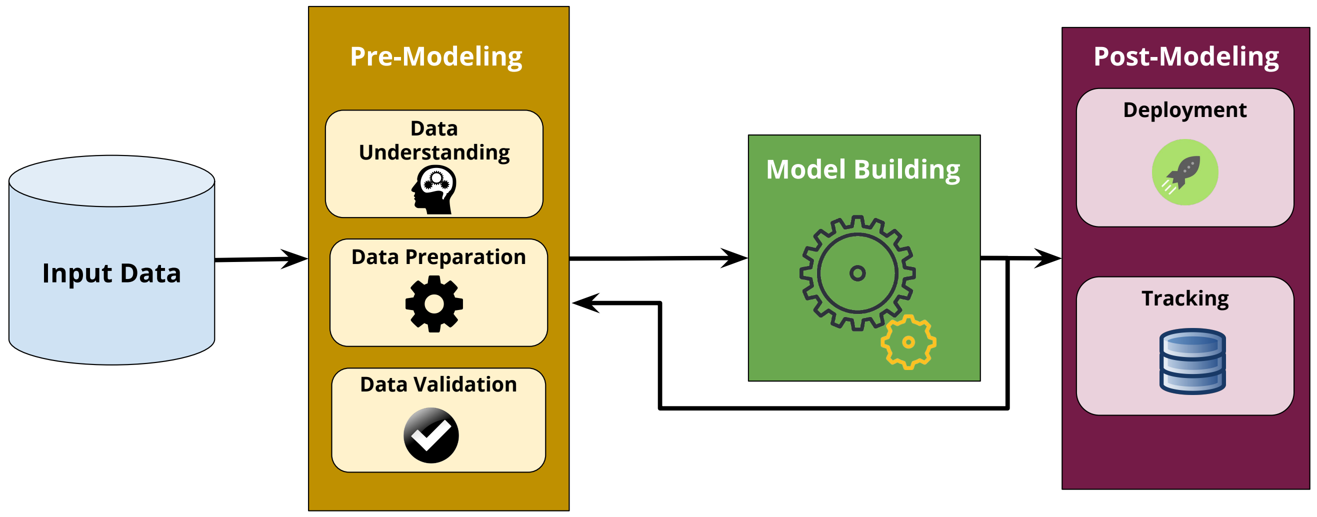

**Figure 8: Machine Learning Production Pipeline**

**Feature Based Analysis**: Data understanding is not limited only to performing a sanity check before model building. It is an iterative process that can occur even after model building to identify the data slices that affect the model quality and behavior. `MLCube` [56] is a popular framework for the feature-based analysis of data understanding. It computes several evaluation metrics and statistics over subsets of data defined by some feature conditions on a machine learning model. `MLCube` integrates its results with an interactive explorer for visualization and models performance comparison. `Smart Drill-Down` [54] has been introduced as an operator that is used to explore, investigate and summarizes group of rows in a relational table. It presents a solution to present the interesting aspects in parts of a table that can affect the model behavior. Smart Drill-Down uses an approximation algorithm to find the optimal list of interesting rules as it is a NP-Hard problem.

**Data Life-cycle Analysis**: Identifying the sources of data errors from a given model and interpreting the dependencies between features of dataset is an important part of data understanding through the whole data life cycle. `GOODs` [46] is a framework that has been introduced to organize structured datasets of different formats at a scale. It extracts metadata from each dataset such as schema, and timestamps to draw the similarity and provenance relations between these datasets. It allows users to find similar datasets, monitor, and annotate them. `ProvDB` [84] is a prototype system that has been built on top of `git` and `Neo4j` [59] graph database with the aim of keeping track of the analyses performed and datasets generated as a meta-data management system that enables collaborative work flows. `ProvDB` allows data scientists to query the captured information. Also, it can help in identifying flaws that can be extracted from data science process itself in addition to automatic monitor, and analysis of deployed models. Ground [47] is an open-source data context service that manages the data storage which facilitates the good use of data. This system was named as `Ground` to indicate their target of unifying the ground of data scientists by integrating some underground services like crawlers, version storage, search index, scheduling, and work flow. Moreover, there are some above ground applications like model serving, reproducibility, analytics and visualization tools, information extractors and security auditing. This unified ground will help to create a data context layer in the big data stack. It is a community effort that can provide useful open source where several applications, and services can be easily integrated and plugged into the same ground.

### *6.1.2 Data Validation*

Data validation is the block that separates the data preparation from model training in the machine learning production pipeline. Several aspects are considered in this step such as adding new features, cleaning the existing ones before building or updating the model, and automatically inserting corrections to invalid data.

**Automatic Data invalidity Diagnosis and correction**: In general, problems may occur in datasets, especially incremental ones, that affect its coherency. For example, some data labels go from capital to lower case, some feature has various currency values, or person age written in different formats as number of years or months. This data invalidity requires automatic fixation at the time of insertion. `Data X-Ray` [127] has been introduced as a diagnostic tool for data systematic errors that are inherited from the process of producing the data itself. It is implemented over concept of MapReduce to allow its scalability. The main contributions of this tool are designing a fast algorithm for diagnosing data on a large-scale, using Bayesian analysis to formulate a model that defines the good diagnosis principles, and transforming the normal diagnosis problem into a search task for finding the common properties of the erroneous features. Experiments made on this tool outperforms the alternative techniques like feature selection algorithms and showed that it can effectively identify the causes of data errors. `MacroBase`[3] is an open source framework for data analytics, and search engine for big, and fast data streams. It enables modular, fast and accurate analysis that can de-

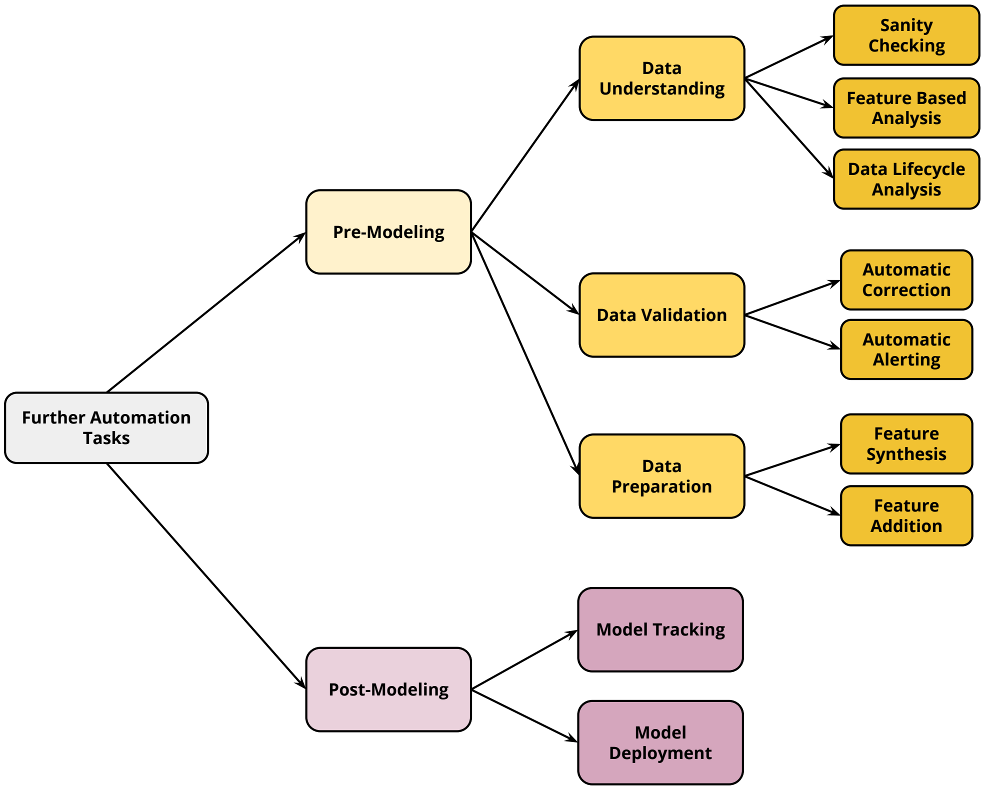

**Figure 9: Further automation tasks that ease the data scientist work.**

tect and monitor unusual behavior in data and deliver summaries of important landmarks over the data streams that represent the unusual behavior. `MacroBase` achieves a speed up to 2 million events per second for each query on one core. The main contribution of `MacroBase` is its ability to optimize the combination of explanation and classification tasks for fast data streams.

**Alert Combining**: When there are multiple alerts, the system should be able to relate them together and determine the root cause of these alerts to facilitate the process of automatic repair by combining multiple alerts into a few ones. [17] proposes a framework that applies different techniques from record-linkage to the search of good repairs by introducing an approach defined with two greedy algorithms with a cubic time complexity in the database size. In addition, some optimizations are added if there is any presence of duplicate instances or records that greatly improve the performance and scalability. Experimental results on this framework showed a great improvement in the performance with little cost of the repair quality. `Conflict Hypergraph` [63] has been introduced as an approximation algorithm which can be used to clean, and repair inconsistencies of a fixed set of functional dependencies with the minimum number of modifications in databases. This algorithm tries to find a solution that is far from the optimum repair which is NP-Hard, with distance less than a fixed constant factor.

### *6.1.3 Data Preparation*

Data preparation is considered as the most time consuming stage in the pipeline due to the presence of many various data preprocessing algorithms including Normalization, Bucketization, Winsorizing, One-Hot encoding, Feature Crosses, etc. In addition, new features can be synthesized from current available data that are better representatives for patterns in data and eases the role of the modeling process. Solutions in the literature either merge the feature preprocessing algorithms with the model training phase as a top layer over algorithm selection that needs to be optimized too. However, some other solutions depends on different types of auto encoder, Restricted Boltzmann Machines (RBM), [40] by feeding the data features directly to a deep neural networks that can figure out the best features which can be generated using stochastic artificial neural networks that can learn the probability distribution over the set of data inputs.

Another possibility for model performance improvement is adding more features to the dataset. However, finding suitable and complementary data is a very difficult task. Thus, many trials have been made to create a repositories for the wide range of datasets in different domains. Recently, Google has initiated a search engine for datasets[30]. In addition, `openML`[124] is a well-organized data repository that allow users to find different datasets, data analysis flows, explore results of these flows, and create tasks that could be shared with the data scientists community and ease the pro-

[30]Google Dataset Search `https://toolbox.google.com/datasetsearch`

cess of finding a solution. Furthermore, in practice, there is a need for version control system for datasets which is offered by `DataHub` [14] that can keep track of different datasets versions, and allows collaborative incremental work on data or roll backs in case of error occurrences. In particular, `Datahub` is a web client for dataset version control system like git, used for easy manipulation of datasets. It provides a complete ecosystem data ingestion, processing, repairing, integration, discovery, query, visualization, and analytics with a Restful API. It allows a user to share data, collaborate with others, and perform a wide range of operations from the available suite of tools to process the data. In addition, there several attempts for automatic feature synthesis from available data features. For instance, *Feature Tools* [58] is a library for automated feature engineering which follows a deep feature synthesis algorithm that can work with relational databases making use of the entity, forward, and backward relations between tables in generating new higher level features iteratively that can improve the modeling performance.

### 6.2 Post-Modeling

In practice, there is an urgent need to try the integration of best algorithms, and tools in different pipeline phases in a single workflow. This step will be the corner-stone of data scientist replacement. Recently, `mlFlow` has been introduced as an open source platform to manage the machine learning pipeline from end-to-end. It is a language agnostic platform that has a REST API, and Command-Line interface in addition to APIs for most popular programming languages like Python, R, and Java. The `mlFlow` performs three different operations which are:

- Record results from experiments and work flows made by different tools and algorithms. In addition, code versions with metrics used, parameter configurations, and visualizations made can all be tracked and stored.
- Package the code used in a chain of reusable and reproducible format to be shared with all the community or to be transferred directly to production. Over and above, it handles all the needed dependencies, and entry points.
- Manage and Deploy the models built from the different work flows over wide range of platforms.

`mlFlow` facilitates many tasks for the data scientist. However, it still lacks the smartness of recommending best work flows that are suitable for each task, and requires human interaction in taking several actions and solving conflicts that occur by transferring models between different platforms.

Similarly, *ModelChimp*[31] provides a server based solution for tracking machine learning, and deep learning experiments that can be connected to an external PostgreSQL database for easily storage and retrieval of results. In addition, it supports real-time visualization for tracking the training process with different metrics, and parameters. *ModelChimp* supports most popular frameworks such as `scikit-learn`, `Keras`, `PyTorch`, and `TensorFlow`. Additionally, *datmo*[32] is an open source tool, in Python, for production model management that helps data scientists to store experiments logs and results with easy reproducibility, and project versioning. Moreover, it allows synchronization between these stored logs with user private cloud storage folders.

[31]`https://modelchimp.com/`

[32]`https://github.com/datmo/datmo`

## 7. OPEN CHALLENGES AND FUTURE DIRECTIONS

Although in the last years, there has been increasing research efforts to tackle the challenges of the automated machine learning domain, however, there are still several open challenges and research directions that needs to be tackled to achieve the ultimate goals and vision of the AutoML domain. In this section, we highlight some of these challenges that need to be tackled to improve the state-of-the-art.

**Scalability**: In practice, a main limitation of the centralized frameworks for automating the solutions for the CASH problem (e.g., `Auto-Weka`, `Auto-Sklearn`) is that they are tightly coupled with a machine learning library (e.g., `Weka`, `scikit-learn`, `R`) that can only work on a *single* node which makes them not applicable in the case of large data volumes. In practice, as the scale of data produced daily is increasing continuously at an exponential scale, several distributed machine learning platforms have been recently introduced. Examples include `Spark MLib` [83], `Mahout`[33] and `SystemML` [16]. Although there have been some initial efforts for distributed automated framework for the CASH problem. However, the proposed distributed solutions are still simple and limited in their capabilities. More research efforts and novel solutions are required to tackle the challenge of automatically building and tuning machine learning models over massive datasets.

**Optimization Techniques**: In practice, different AutoML frameworks use different techniques for hyper-parameter optimization of the machine learning algorithms. For instance, `Auto-Weka` and `Auto-Sklearn` use the SMAC technique with cross-fold validation during the hyper-parameter configuration optimization and evaluation. On the other hand, `ML-Plan` uses the hierarchical task network with Monte Carlo Cross-Validation. Other tools, including `Recipe` [27] and `TPOT`, use genetic programming, and pareto optimization for generating candidate pipelines. In practice, it is difficult to find a clear winner or one-size-fits-all technique. In other words, there is no single method that will be able to outperform all other techniques on the different datasets with their various characteristics, types of search spaces and metrics (e.g., time and accuracy). Thus, there is a crucial need to understand the Pros and Cons of these optimization techniques so that AutoML systems can automatically tune their hyper-parameter optimization techniques or their strategy for exploring and traversing the search space. Such decision automation should provide improved performance over picking and relying on a fixed strategy. Similarly, for the various introduced meta-learning techniques, there is no clear systematic process or evaluation metrics to quantitatively assess and compare the impact of these techniques on reducing the search space. Recently, some competitions

[33]`https://mahout.apache.org/`

and challenges[34,35] have been introduced and organized to address this issue such as the DARPA D3M Automatic Machine Learning competition [107].

**Time Budget**: A common important parameter for AutoML systems is the user *time budget* to wait before getting the recommended pipeline. Clearly, the bigger the time budget, the more the chance for the AutoML system to explore various options in the search space and the higher probability to get a better recommendation. However, the bigger time budget used, the longer waiting time and the higher computing resource consumption, which could be translated into a higher monetary bill in the case of using cloud-based resources. On the other hand, a small-time budget means a shorter waiting time but a lower chance to get the best recommendation. However, it should be noted that increasing the time budget from $X$ to $2X$ does not necessarily lead to a big increase on the quality of the results of the recommended pipeline, if any at all. In many scenarios, this extra time budget can be used for exploring more of the unpromising branches in the search space or exploring branches that have very little gain, if any. For example, the accuracy of the returned models from running the `AutoSklearn` framework over the `Abalone` dataset[36] with time budgets of 4 hours and 8 hours are almost the same (25%). Thus, accurately estimating or determining the adequate time budget to optimize this trade-off is another challenging decision that can not be done by non-expert end users. Therefore, it is crucial to tackle such challenge by automatically predicting/recommending the adequate time budget for the modeling process. The `VDS` [107] framework provided a first attempt to tackle this challenge by proposing an interactive approach that relies on meta learning to provide a quick first model recommendation that can achieve a reasonable quality while conducting an offline optimization process and providing the user with a *stream* of models with better accuracy. However, more research efforts to tackle this challenge are still required.

**Composability** Nowadays, several machine learning solutions (e.g., `Weka`, `Scikit-Learn`, `R`, `MLib`, `Mahout`) have become popular. However, these ML solutions significantly vary in their available techniques (e.g., learning algorithms, preprocessors, and feature selectors) to support each phase of the machine learning pipeline. Clearly, the quality of the machine learning pipelines that can be produced by any of these platforms depends on the availability of several techniques/algorithms that can be utilized in each step of the pipeline. In particular, the more available techniques/algorithms in a machine learning platform, the higher the ability and probability of producing a well-performing machine learning pipeline. In practice, it is very challenging to have optimized implementations for all of the algorithms/techniques of the different steps of the machine learning pipeline available in a single package, or library. The `ML-Plan` framework [89] has been attempting to tackle the composability challenge on building machine learning pipelines. In particular, it integrates a superset of both `Weka` and `Scikit-Learn` algorithms to construct a full pipeline. The initial results of this approach have shown that the composable pipelines over `Weka` and `Scikit-Learn` do not significantly outperform the outcomes from `Auto-Weka` and `Auto-Sklearn` frameworks especially with big datasets and small time budgets. However, we believe that there are several reasons behind these results. First, combining the algorithms/techniques of more than one machine learning platform causes a dramatic increase in the search space. Thus, to tackle this challenge, there is a crucial need for a smart and efficient search algorithm that can effectively reduce the search space and focus on the promising branches. Using meta-learning approaches can be an effective solution to tackle this challenge. Second, combining services from more than one framework can involve a significant overhead for the data and message communications between the different frameworks. Therefore, there is a crucial need for a smart *cost-based* optimizer that can accurately estimate the gain and cost of each recommended composed pipeline and be able to choose the composable recommendations when they are able to achieve a clear performance gain. Third, the `ML-Plan` has been combining the services of two single node machine learning services (`Weka` and `Scikit-Learn`). We believe that the best gain of the composability mechanism will be achieved by combining the performance power of distributed systems (e.g., `MLib`) with the rich functionality of many centralized systems.

**User friendliness**: In general, most of the current tools and framework can not be considered to be user friendly. They still need sophisticated technical skills to be deployed and used. Such challenge limits its usability and wide acceptance among layman users and domain experts (e.g., physicians, accountants) who commonly have limited technical skills. Providing an interactive and light-weight web interfaces for such framework can be one of the approaches to tackle these challenges.

**Continuous delivery pipeline**: Continuous delivery is defined as creating a repeatable, reliable and incrementally improving process for taking software from concept to customer. Integrating machine learning models into continuous delivery pipelines for productive use has not recently drawn much attention, because usually the data scientists push them directly into the production environment with all the drawbacks this approach may have, such as no proper unit and integration testing.

**Data Validation**: In this context, most of the solutions in literature focus on problem detection and user notification only. However, automatic correction hasn't been investigated in a good manner that covers several possible domains of datasets and reduce the data scientist's role in machine learning production pipeline. In addition, as the possible data repairing is a NP-Hard problem, there is a need to find more approximation techniques that can solve this problem.

**Data Preparation**: In practice, there is a crucial need for automating the feature extraction process as it is considered as one of the most time consuming part of the pipeline. In practice, most of the systems neglect the automation of transferring data features into different domain space like performing principal component analysis, or linear discriminant analysis and when they improve the model performance. In addition, we believe that there is a room for

[34] `https://www.4paradigm.com/competition/nips2018`
[35] `http://automl.chalearn.org/`
[36] `https://www.openml.org/d/183`

improvement of current auto-encoders types like restricted Boltzmann Machines. So, further research is needed to try different architectures and interpret them to have the ability to automate the choice of suitable encoders. Furthermore, there are various techniques for measuring a score for the feature importance which is a very important part to automate the feature selection process. However, there is no comprehensive comparative studies between these methods or good recipes that can recommend when to use each of these techniques.

**Model Deployment and Life Cycle**: Recently, there some tools and frameworks that have been introduced to ease the data scientist work and automate the machine learning production. However, in practice, there is still a need to integrate these different systems along the whole pipeline. For example, there is a large room for improvement regarding the automatic choice of the good work flows specific to each problem and how to integrate more data understanding, validation and preparation techniques with the work flows. In particular, these frameworks are still not providing the end-user with any *smartness* in the decision making process which is a corner stone towards replacing the role of human in the loop.

## 8. CONCLUSION

Machine learning has become one of the main engines of the current era. The production pipeline of a machine learning models passe through different phases and stages that require wide knowledge of several available tools, and algorithms. However, as the scale of data produced daily is increasing continuously at an exponential scale, it has become essential to automate this process. In this survey, we have covered comprehensively the state-of-the-art research effort in the domain of AutoML frameworks. We have also highlighted research directions and open challenges that need to be addressed in order to achieve the vision and goals of the AutoML process. We hope that our survey serves as a useful resource for the community, for both researchers and practitioners, to understand the challenges of the domain and provide useful insight for further advancing the state-of-the-art in several directions.

## Acknowledgment

This work of Sherif Sakr and Mohamed Maher is funded by the European Regional Development Funds via the Mobilitas Plus programme (grant MOBTT75). The work of Radwa Elshawi is funded by the European Regional Development Funds via the Mobilitas Plus programme (MOBJD341).

## 9. REFERENCES

[1] Karim Ahmed and Lorenzo Torresani. Maskconnect: Connectivity learning by gradient descent. *Lecture Notes in Computer Science*, page 362âĂŞ378, 2018.

[2] Peter J Angeline, Gregory M Saunders, and Jordan B Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE transactions on Neural Networks*, 5(1):54–65, 1994.

[3] Peter Bailis, Edward Gan, Samuel Madden, Deepak Narayanan, Kexin Rong, and Sahaana Suri. Macrobase: Prioritizing attention in fast data. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 541–556, New York, NY, USA, 2017. ACM.

[4] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[5] Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michele Sebag. Collaborative hyperparameter tuning. In *International conference on machine learning*, pages 199–207, 2013.

[6] Jonathan Baxter. *Learning internal representations.* Flinders University of S. Aust., 1995.

[7] Richard E Bellman. *Adaptive control processes: a guided tour*, volume 2045. Princeton university press, 2015.

[8] Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 17–36, 2012.

[9] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[10] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[11] James Bergstra, Dan Yamins, and David D Cox. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, pages 13–20. Citeseer, 2013.

[12] James Bergstra, Daniel Yamins, and David Daniel Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. 2013.

[13] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

[14] Anant P. Bhardwaj, Souvik Bhattacherjee, Amit Chavan, Amol Deshpande, Aaron J. Elmore, Samuel Madden, and Aditya G. Parameswaran. Datahub: Collaborative data science & dataset version management at scale. *CoRR*, abs/1409.0798, 2014.

[15] Besim Bilalli, Alberto Abelló, and Tomas Aluja-Banet. On the predictive power of meta-features in openml. *International Journal of Applied Mathematics and Computer Science*, 27(4):697–712, 2017.

[16] Matthias Boehm, Michael W Dusenberry, Deron Eriksson, Alexandre V Evfimievski, Faraz Makari Manshadi, Niketan Pansare, Berthold Reinwald, Frederick R Reiss, Prithviraj Sen, Arvind C Surve, et al. Systemml: Declarative machine learning on spark. *Proceedings of the VLDB Endowment*, 9(13):1425–1436, 2016.

[17] Philip Bohannon, Wenfei Fan, Michael Flaster, and Rajeev Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pages 143–154, New York, NY, USA, 2005. ACM.

[18] Pavel Brazdil, Christophe Giraud Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: Applications to data mining*. Springer Science & Business Media, 2008.

[19] Pavel B Brazdil, Carlos Soares, and Joaquim Pinto Da Costa. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277, 2003.

[20] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[21] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[22] Rich Caruana. Learning many related tasks at the same time with backpropagation. In *Advances in neural information processing systems*, pages 657–664, 1995.

[23] Boyuan Chen, Harvey Wu, Warren Mo, Ishanu Chattopadhyay, and Hod Lipson. Autostacker: A compositional evolutionary learning system. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, pages 402–409, New York, NY, USA, 2018. ACM.

[24] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8609–8613. IEEE, 2013.

[25] Lawrence Davis. Handbook of genetic algorithms. 1991.

[26] Alex GC de Sá, Alex A Freitas, and Gisele L Pappa. Automated selection and configuration of multi-label classification algorithms with grammar-based genetic programming. In *International Conference on Parallel Problem Solving from Nature*, pages 308–320. Springer, 2018.

[27] Alex GC de Sá, Walter José GS Pinto, Luiz Otavio VB Oliveira, and Gisele L Pappa. Recipe: a grammar-based framework for automatically evolving classification pipelines. In *European Conference on Genetic Programming*, pages 246–261. Springer, 2017.

[28] Alex Guimarães Cardoso de Sá, Walter José G. S. Pinto, Luiz Otávio Vilas Boas Oliveira, and Gisele L. Pappa. RECIPE: A grammar-based framework for automatically evolving classification pipelines. In *EuroGP*, volume 10196 of *Lecture Notes in Computer Science*, pages 246–261, 2017.

[29] Sander Dieleman, Jan Schlter, Colin Raffel, Eben Olson, Sren Kaae Snderby, Daniel Nouri, Daniel Maturana, Martin Thoma, Eric Battenberg, Jack Kelly, et al. Lasagne: First release., august 2015. *URL http://dx. doi. org/10.5281/zenodo*, 27878, 2016.

[30] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[31] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655, 2014.

[32] Patrícia Maforte dos Santos, Teresa Bernarda Ludermir, and Ricardo Bastos Cavalcante Prudencio. Selection of time series forecasting models based on performance information. In *Fourth International Conference on Hybrid Intelligent Systems (HIS'04)*, pages 366–371. IEEE, 2004.

[33] Iddo Drori, Yamuna Krishnamurthy, Remi Rampin, Raoni de Paula Lourenco, Jorge Piazentin Ono, Kyunghyun Cho, Claudio Silva, and Juliana Freire. Alphad3m: Machine learning pipeline synthesis. In *AutoML Workshop at ICML*, 2018.

[34] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[35] Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger Hoos, and Kevin Leyton-Brown. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, volume 10, page 3, 2013.

[36] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. 2018.

[37] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. *arXiv preprint arXiv:1807.01774*, 2018.

[38] M Giselle Fernández-Godino, Chanyoung Park, Nam-Ho Kim, and Raphael T Haftka. Review of multi-fidelity models. *arXiv preprint arXiv:1609.07196*, 2016.

[39] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 2755–2763, Cambridge, MA, USA, 2015. MIT Press.

[40] Asja Fischer and Christian Igel. Training restricted boltzmann machines. *Pattern Recogn.*, 47(1):25–39, January 2014.

[41] Brian Fitzgerald. Software crisis 2.0. *Computer*, 45(4), 2012.

[42] Johannes Fürnkranz and Johann Petrak. An evaluation of landmarking variants. In *Working Notes of the ECML/PKDD 2000 Workshop on Integrating Aspects of Data Mining, Decision Support and Meta-Learning*, pages 57–68, 2001.

[43] Nicolo Fusi, Rishit Sheth, and Huseyn Melih Elibol. Probabilistic matrix factorization for automated machine learning. *arXiv preprint arXiv:1705.05355*, 2017.

[44] Christophe Giraud-Carrier. Metalearning-a tutorial. In *Tutorial at the 7th international conference on machine learning and applications (ICMLA), San Diego, California, USA*, 2008.

[45] Silvio B Guerra, Ricardo BC Prudêncio, and

Teresa B Ludermir. Predicting the performance of learning algorithms using support vector machines as meta-regressors. In *International Conference on Artificial Neural Networks*, pages 523–532. Springer, 2008.

[46] Alon Halevy, Flip Korn, Natalya F. Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. Goods: Organizing google's datasets. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 795–806, New York, NY, USA, 2016. ACM.

[47] Joseph M. Hellerstein, Vikram Sreekanti, Joseph E. Gonzalez, James Dalton, Akon Dey, Sreyashi Nag, Krishna Ramachandran, Sudhanshu Arora, Arka Bhattacharyya, Shirshanka Das, Mark Donsky, Gabriel Fierro, Chang She, Carl Steinbach, Venkat Subramanian, and Eric Sun. Ground: A data context service. In *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*, 2017.

[48] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: Closing the generalization gap in large batch training of neural networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 1729–1739, USA, 2017. Curran Associates Inc.

[49] John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[50] Yi-Qi Hu, Yang Yu, Wei-Wei Tu, Qiang Yang, Yuqiang Chen, and Wenyuan Dai. Multi-fidelity automatic hyper-parameter tuning via transfer series expansion. 2019.

[51] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

[52] Kevin G Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *AISTATS*, pages 240–248, 2016.

[53] Haifeng Jin, Qingquan Song, and Xia Hu. Efficient neural architecture search with network morphism. *CoRR*, abs/1806.10282, 2018.

[54] Manas Joglekar, Hector Garcia-Molina, and Aditya Parameswaran. Smart drill-down: A new data exploration operator. *Proc. VLDB Endow.*, 8(12):1928–1931, August 2015.

[55] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[56] Minsuk Kahng, Dezhi Fang, and Duen Horng (Polo) Chau. Visual exploration of machine learning results using data cube analysis. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, HILDA '16, pages 1:1–1:6, New York, NY, USA, 2016. ACM.

[57] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric Xing. Neural architecture search with bayesian optimisation and optimal transport, 2018.

[58] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015, Paris, France, October 19-21, 2015*, pages 1–10. IEEE, 2015.

[59] Chris Kemper. *Beginning Neo4J*. Apress, Berkely, CA, USA, 1st edition, 2015.

[60] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[61] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[62] Aaron Klein, Eric Christiansen, Kevin Murphy, and Frank Hutter. Towards reproducible neural architecture and hyperparameter search. 2018.

[63] Solmaz Kolahi and Laks V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *Proceedings of the 12th International Conference on Database Theory*, ICDT '09, pages 53–62, New York, NY, USA, 2009. ACM.

[64] Brent Komer, James Bergstra, and Chris Eliasmith. Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In *ICML workshop on AutoML*, pages 2825–2830. Citeseer, 2014.

[65] Christian Köpf and Ioannis Iglezakis. Combination of task description strategies and case base properties for meta-learning. In *Proceedings of the 2nd international workshop on integration and collaboration aspects of data mining, decision support and meta-learning*, pages 65–76, 2002.

[66] Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *J. Mach. Learn. Res.*, 18(1):826–830, January 2017.

[67] Tim Kraska, Ameet Talwalkar, John C Duchi, Rean Griffith, Michael J Franklin, and Michael I Jordan. Mlbase: A distributed machine-learning system. In *Cidr*, volume 1, pages 2–1, 2013.

[68] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[69] Harold J Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86(1):97–106, 1964.

[70] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search, 2019.

[71] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560*, 2016.

[72] Jason Liang, Elliot Meyerson, Babak Hodjat, Dan Fink, Karl Mutch, and Risto Miikkulainen. Evolutionary neural automl for deep learning, 2019.

[73] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[74] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017.

[75] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[76] Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. Nsga-net: a multi-objective genetic algorithm for neural architecture search. *arXiv preprint arXiv:1810.03522*, 2018.

[77] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in neural information processing systems*, pages 7816–7827, 2018.

[78] Mohamed Maher and Sherif Sakr. Smartml: A meta learning-based framework for automated selection and hyperparameter tuning for machine learning algorithms. In *EDBT: 22nd International Conference on Extending Database Technology*, 2019.

[79] Andrew March and Karen Willcox. Provably convergent multifidelity optimization algorithm not requiring high-fidelity derivatives. *AIAA journal*, 50(5):1079–1089, 2012.

[80] Ruben Martinez-Cantin. Bayesopt: A bayesian optimization library for nonlinear optimization, experimental design and bandits. *The Journal of Machine Learning Research*, 15(1):3735–3739, 2014.

[81] Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

[82] Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Towards automatically-tuned neural networks. In *Workshop on Automatic Machine Learning*, pages 58–65, 2016.

[83] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016.

[84] Hui Miao, Amit Chavan, and Amol Deshpande. Provdb: A system for lifecycle management of collaborative analysis workflows. *CoRR*, abs/1610.04963, 2016.

[85] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier, 2019.

[86] Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. Designing neural networks using genetic algorithms. In *ICGA*, volume 89, pages 379–384, 1989.

[87] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. *arXiv preprint arXiv:1707.03141*, 2017.

[88] Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of bayesian methods for seeking the extremum. *Towards global optimization*, 2(117-129):2, 1978.

[89] Felix Mohr, Marcel Wever, and Eyke Hüllermeier. Ml-plan: Automated machine learning via hierarchical planning. *Machine Learning*, 107(8-10):1495–1515, 2018.

[90] Douglas C Montgomery. *Design and analysis of experiments*. John wiley & sons, 2017.

[91] Yu Nesterov. A method of solving a convex programming problem with convergence rate $o(1/k^2)$ $o(1/k2)$. In *Sov. Math. Dokl*, volume 27.

[92] Eleni Nisioti, K Chatzidimitriou, and A Symeonidis. Predicting hyperparameters from meta-features in binary classification problems. In *AutoML Workshop at ICML*, 2018.

[93] Randal S. Olson and Jason H. Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors, *Proceedings of the Workshop on Automatic Machine Learning*, volume 64 of *Proceedings of Machine Learning Research*, pages 66–74, New York, New York, USA, 24 Jun 2016. PMLR.

[94] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[95] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[96] Martin Pelikan, David E Goldberg, and Erick Cantú-Paz. Boa: The bayesian optimization algorithm. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1*, pages 525–532. Morgan Kaufmann Publishers Inc., 1999.

[97] Valerio Perrone, Rodolphe Jenatton, Matthias Seeger, and Cedric Archambeau. Multiple adaptive bayesian linear regression for scalable bayesian optimization with warm start. *arXiv preprint arXiv:1712.02902*, 2017.

[98] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[99] Elijah Polak. *Optimization: algorithms and consistent approximations*, volume 124. Springer Science & Business Media, 2012.

[100] Philipp Probst and Anne-Laure Boulesteix. To tune or not to tune the number of trees in random forest. *Journal of Machine Learning Research*, 18:181–1, 2017.

[101] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[102] Jesse Read, Peter Reutemann, Bernhard Pfahringer, and Geoff Holmes. Meka: a multi-label/multi-target extension to weka. *The Journal of Machine Learning Research*, 17(1):667–671, 2016.

[103] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.

[104] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2902–2911. JMLR. org, 2017.

[105] Matthias Reif, Faisal Shafait, Markus Goldstein, Thomas Breuel, and Andreas Dengel. Automatic classifier selection for non-experts. *Pattern Analysis and Applications*, 17(1):83–96, 2014.

[106] Sherif Sakr and Albert Y. Zomaya, editors. *Encyclopedia of Big Data Technologies*. Springer, 2019.

[107] Zeyuan Shang, Emanuel Zgraggen, Benedetto Buratti, Ferdinand Kossmann, Yeounoh Chung, Philipp Eichmann, Carsten Binnig, Eli Upfal, and Tim Kraska. Democratizing data science through interactive curation of ml pipelines. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2019.

[108] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813, 2014.

[109] Richard Shin, Charles Packer, and Dawn Song. Differentiable neural network architecture search. 2018.

[110] Tarique Siddiqui, Albert Kim, John Lee, Karrie Karahalios, and Aditya Parameswaran. Effortless data exploration with zenvisage: An expressive and interactive visual analytics system. *Proc. VLDB Endow.*, 10(4):457–468, November 2016.

[111] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087, 2017.

[112] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[113] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180, 2015.

[114] Carlos Soares, Pavel B Brazdil, and Petr Kuba. A meta-learning method to select the kernel width in support vector regression. *Machine learning*, 54(3):195–209, 2004.

[115] Evan R Sparks, Ameet Talwalkar, Daniel Haas, Michael J Franklin, Michael I Jordan, and Tim Kraska. Automating model search for large scale machine learning. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, pages 368–380. ACM, 2015.

[116] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

[117] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[118] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[119] Thomas Swearingen, Will Drevo, Bennett Cyphers, Alfredo Cuesta-Infante, Arun Ross, and Kalyan Veeramachaneni. Atm: A distributed, collaborative, scalable system for automated machine learning. pages 151–162, 12 2017.

[120] Kevin Swersky, David Duvenaud, Jasper Snoek, Frank Hutter, and Michael A Osborne. Raiders of the lost architecture: Kernels for bayesian optimization in conditional parameter spaces. *arXiv preprint arXiv:1409.4011*, 2014.

[121] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Mining multi-label data. In *In Data Mining and Knowledge Discovery Handbook*, pages 667–685, 2010.

[122] Thanasis Vafeiadis, Konstantinos I Diamantaras, George Sarigiannidis, and K Ch Chatzisavvas. A comparison of machine learning techniques for customer churn prediction. *Simulation Modelling Practice and Theory*, 55:1–9, 2015.

[123] Joaquin Vanschoren. Meta-learning: A survey. *CoRR*, abs/1810.03548, 2018.

[124] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

[125] Manasi Vartak, Sajjadur Rahman, Samuel Madden, Aditya G. Parameswaran, and Neoklis Polyzotis. Seedb: Efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 8(13):2182–2193, 2015.

[126] Wei Wang, Sheng Wang, Jinyang Gao, Meihui Zhang, Gang Chen, Teck Khim Ng, and Beng Chin Ooi. Rafiki: Machine learning as an analytics service system. *CoRR*, abs/1804.06087, 2018.

[127] Xiaolan Wang, Xin Luna Dong, and Alexandra Meliou. Data x-ray: A diagnostic tool for data errors. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 1231–1245, New York, NY, USA, 2015. ACM.

[128] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[129] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Scalable gaussian process-based transfer surrogates for hyperparameter optimization. *Machine Learning*, 107(1):43–78, 2018.

[130] Chengrun Yang, Yuji Akimoto, Dae Won Kim, and Madeleine Udell. OBOE: Collaborative filtering for AutoML initialization. *arXiv preprint arXiv:1808.03233*, 2019.

[131] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[132] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[133] Zheguang Zhao, Lorenzo De Stefani, Emanuel Zgraggen, Carsten Binnig, Eli Upfal, and Tim Kraska. Controlling false discoveries during interactive data exploration. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 527–540, New York, NY, USA, 2017. ACM.

[134] AG Zhilinskas. Single-step bayesian search method for an extremum of functions of a single variable. *Cybernetics and Systems Analysis*, 11(1):160–166, 1975.

[135] Albert Y Zomaya and Sherif Sakr. *Handbook of big data technologies*. Springer, 2017.

[136] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[137] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.