AutoML to Date and Beyond: Challenges and Opportunities

SHUBHRA KANTI KARMAKER ("SANTU"), Auburn University (Previously MIT LIDS)
MD. MAHADI HASSAN, Auburn University
MICAH J. SMITH and LEI XU, MIT LIDS
CHENGXIANG ZHAI, University of Illinois Urbana Champaign
KALYAN VEERAMACHANENI, MIT LIDS

As big data becomes ubiquitous across domains, and more and more stakeholders aspire to make the most of their data, demand for machine learning tools has spurred researchers to explore the possibilities of automated machine learning (AutoML). AutoML tools aim to make machine learning accessible for non-machine learning experts (domain experts), to improve the efficiency of machine learning, and to accelerate machine learning research. But although automation and efficiency are among AutoML's main selling points, the process still requires human involvement at a number of vital steps, including understanding the attributes of domain-specific data, defining prediction problems, creating a suitable training dataset, and selecting a promising machine learning technique. These steps often require a prolonged back-and-forth that makes this process inefficient for domain experts and data scientists alike and keeps so-called AutoML systems from being truly automatic. In this review article, we introduce a new classification system for AutoML systems, using a seven-tiered schematic to distinguish these systems based on their level of autonomy. We begin by describing what an end-to-end machine learning pipeline actually looks like, and which subtasks of the machine learning pipeline have been automated so far. We highlight those subtasks that are still done manually—generally by a data scientist—and explain how this limits domain experts' access to machine learning. Next, we introduce our novel level-based taxonomy for AutoML systems and define each level according to the scope of automation support provided. Finally, we lay out a roadmap for the future, pinpointing the research required to further automate the end-to-end machine learning pipeline and discussing important challenges that stand in the way of this ambitious goal.

CCS Concepts: • **Computing methodologies** → **Machine learning**; **Supervised learning**; • **Information systems** → **Data mining**; **Data analytics;**

Additional Key Words and Phrases: Automated machine learning, interactive data science, democratization of artificial intelligence, predictive analytics

ACM Reference format:
Shubhra Kanti Karmaker ("Santu"), Md. Mahadi Hassan, Micah J. Smith, Lei Xu, ChengXiang Zhai, and Kalyan Veeramachaneni. 2021. AutoML to Date and Beyond: Challenges and Opportunities. *ACM Comput. Surv.* 54, 8, Article 175 (October 2021), 36 pages.
https://doi.org/10.1145/3470918

Authors' addresses: S. K. Karmaker ("Santu") and Md. M. Hassan, Samuel Ginn College of Engineering, 3106 Shelby Center, Auburn University, 345 W Magnolia Ave, Auburn, AL 36849; emails: SKS0086, sibathasan@auburn.edu; M. J. Smith, L. Xu, and K. Veeramachaneni, MIT LIDS, MIT Stata Center, 32 Vassar St, Room 32-D712, Cambridge, MA 02139; emails: {micahs, leix, kalyanv}@mit.edu; C. Zhai, University of Illinois Urbana Champaign, Thomas M. Siebel Center for Computer Science, 201 North Goodwin Avenue MC 258 Urbana, IL 61801-2302; email: czhai@illinois.edu.

0360-0300/2021/10-ART175 $15.00
https://doi.org/10.1145/3470918

1 INTRODUCTION

One of the main consequences of the massive increase in web-based systems has been the proliferation of "big data" automatically generated information that can shed light on user demographics, behaviors, and needs. Companies across domains and applications are increasingly focused on exploiting this data to improve their products and services. The rapid growth of machine learning infrastructure and high-performance computing have significantly contributed to this process. Business entities are hiring more and more data scientists (5× growth) and **machine learning (ML)** engineers (12× growth) to better make sense of their data [4], while both industry and academia have made **artificial intelligence (AI)** and ML research priorities.

As these stakeholders attempt to make the most of their data, demand for machine learning tools has spurred researchers to explore the possibilities of **automated machine learning (AutoML)**. AutoMLis essentially a paradigm for automating the application of machine learning to real-world problems (what we call the "end-to-end ML process"). Although automation and efficiency are among AutoML's main selling points, this process still requires a surprising level of human involvement. To better understand the problem, let us discuss what an end-to-end ML process looks like today.

Generally, a business or enterprise adopts machine learning to fulfill a specific need. For instance, say an organization's leadership team launches a product or service, and customers start interacting with it. Domain experts begin digging deeper into customer behavior, and may deploy sensors and conduct experiments to closely monitor different components of the product. Because all of these transactions, customer interactions, experiments, and sensors produce a huge amount of data every day—too much for these domain experts to manually obtain insights from—data scientists are brought in to help. As domain experts and data scientists are the key users of AutoMLtools, we pause here to define their roles:

- **Domain Expert:** A person who is fluent in the domain where ML is being applied but has minimal knowledge of how ML itself works.
- **Data Scientist:** A person who knows how ML works but has minimal knowledge of the domain where it is being applied.

The data scientist must work with the domain expert to understand the context in which the data was collected, as well as the overall goals. S/he then translates the goal into a computational prediction task, extracts useful features from the raw data [66], and selects a relevant ML model to solve the prediction task. A number of vital steps of this ML process—including understanding the attributes of domain-specific data, defining prediction problems, and creating a suitable training dataset—are generally still done manually by the data scientist on an ad-hoc basis. This often requires a lot of back-and-forth between the data scientist(s) and domain expert(s), making the whole process more difficult and inefficient.

It is only once all of these steps are complete that existing AutoML systems come in—automatically applying and tuning machine learning techniques to the training/testing datasets that have been compiled by the human data scientist, to solve a prediction task that has been carefully defined by the same person. The data scientist is then called upon again, this time to draw valuable insights from the data and models. S/he then reports these results to the domain experts and business leaders, who use them to improve the product and address any issues with it. This whole process is laid out visually in Figure 1.

The term AutoMLis commonly understood to denote an system that can perform the common data science tasks described above with minimal human intervention. However, current AutoML systems with that designation are actually far from achieving that level of automation. We posit

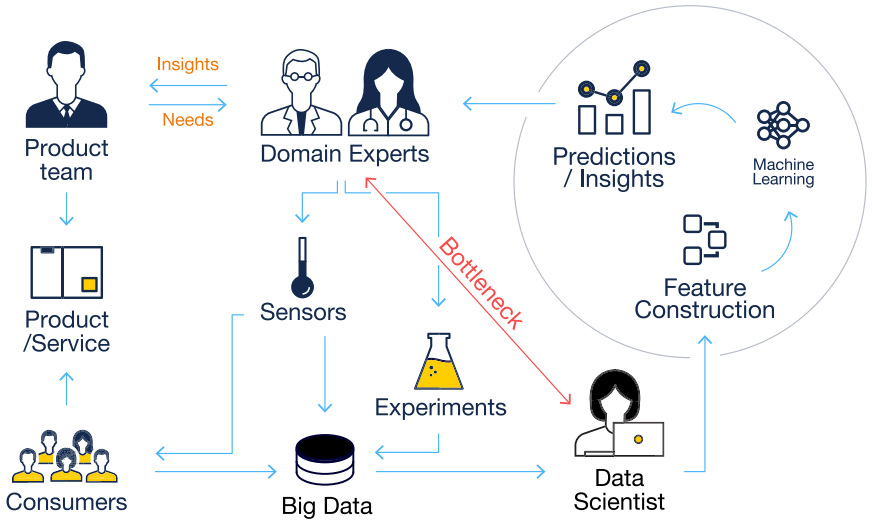

Fig. 1. A typical end-to-end machine learning process, with multiple stakeholders bringing their individual expertise, tools, methods, datasets, and goals.

that the "northstar" goals for AutoML systems should be twofold: to increase the efficiency of data scientists, and to enable domain experts to directly use machine learning using automation tools, inserting their domain knowledge as necessary. The latter goal, if achieved, will bring along a welcome secondary effect—making the powerful tool of machine learning more accessible to non-experts across fields.

How do we achieve this "*northstar*" goal: To achieve this "*northstar*," we must expand the definition of "AutoML" systems, to speak more precisely about these tools and plan for future improvements. In this review article, we present a new level-based classification system for current and future AutoML solutions. We first describe the tasks a data scientist performs during the end-to-end machine learning process (see Section 2), pointing out subtasks that are still generally done manually as well as those that current AutoML systems can perform automatically. We highlight the overall flow, iterative loops within this process, and the sources of bottlenecks (see Section 3). We then present our level-based classification system, which is based on the automation of particular subtasks, and provide examples of existing AutoML systems at each level (see Section 4). Finally, we lay out a road map for the future, pinpointing the research required to further automate the end-to-end machine learning process and discussing important challenges that stand in the way of this ambitious goal (see Section 5).

2 WHAT DOES A DATA SCIENTIST DO?

As shown in the representative end-to-end machine learning process in Figure 1, *data scientists* often perform a well-defined set of functions to achieve analysis goals. We formalize these steps below. Over the past decade, the machine learning and systems communities have worked to automate many, but not all, of these steps.

- **Task Formulation (TF):** In any *data science* endeavor, the *data scientist* first interacts with the domain expert to understand the problem at hand, examines the available data, and formulates a machine learning task to help solve this problem. Formulating the task can

involve a lot of back and forth, where the *data scientist* and domain expert must consider multiple possibilities and check if the required data are available each time before making a decision. This is currently a manual process, and is often long and arduous, impeded by mismatched expectations and communication gaps.

- **Data Visualization, Cleaning, and Curation (DCC):** Throughout the analysis process, data scientists repeatedly return to the original data to propose and revise models and test hypotheses. This often requires cleaning and curating new subsets of data. After identifying the relevant data, data scientists may handle missing values, "*join*" multiple datasets and perform other functions for improving data quality. They may also create visualizations to direct this process and aid in discussions. The *data mining*, *systems*, and *database* communities have progressed appreciably in automating this step, efforts that are well summarized in Ilyas et al. [27] and Chu et al. [11]. In several of these approaches, machine learning itself is used to automatically perform tasks involved in curation, cleaning, and dataset linking. More recent approaches have focused on data cleaning challenges for specific domains, including electronic health records (Miao et al. [44]), time-series data (Zhang et al. [69]), online reviews (Minnich et al. [45]), and wireless sensor networks (Cheng et al. [10]). Another school of effort has primarily focused on interactive or active data cleaning (e.g., Chu et al. [12], Haas et al. [22], and Krishnan et al. [36]). For a more detailed review of existing literature on data visualization, cleaning, and curation, refer to Appendix A.
- **Prediction Engineering (PE):** Prediction engineering involves constructing and assigning labels to data points according to the set prediction task, and creating meaningful training and testing sets as collections of `<data_point, label>` tuples. This task is usually done manually by data scientists, sometimes with the help of human data annotators. However, there is a growing demand for automated *prediction engineering*. Kanter et al. [29] tried to automate this process (the first attempt of this kind) by proposing a generalized three-part framework "Label, Segment, Featurize," which provides abstractions that enable data scientists to customize the *prediction engineering* process for unique prediction problems.
- **Feature Engineering (FE):** By the end of the first three steps, a data scientist will have defined a ML task (most commonly a predictive task), created training and testing sets, and begun the model building process—the first step of which is to attempt to construct informative features from raw data. This process is called "feature engineering," which we define formally as follows.

  Definition 2.1. Feature engineering is the process of transforming raw data into features that can better represent an underlying problem for computational predictive models, resulting in improved model accuracy on unseen data.

  Later, these features can be directly fed to ML models to train them and make predictions. In the past 5 years, a number of efforts have focused on automating the process of *feature engineering* (Kanter and Veeramachaneni [30], Katz et al. [32], Kaul et al. [33], Khurana et al. [34], Mountantonakis and Tzitzikas [46], van den Bosch [62]). There are now multiple systems and open source libraries that focus on extracting useful features from data and generating feature matrices to construct the training set. For more details on these efforts to automate *feature engineering*, refer to Appendix B.
- **Machine Learning:** Once feature engineering is complete, the data scientist selects a suitable machine learning technique based on the prediction task, and begins training models. These models consist of basic machine learning techniques—such as decision trees, support vectors machines, linear regressions, neural networks, and so on—which have current implementations across various software packages including scikit-learn [49]

and weka [67]. One crucial task associated with training is optimizing, or *tuning*, any hyperparameters associated with the learning model.

- **Hyperparameter Tuning:** Machine learning models often contain multiple hyperparameters, the values of which are critical for obtaining good performance. Data scientists must tune these hyperparameters to find models that work reasonably well. Efforts to automate tuning include Bengio [3], Bergstra and Bengio [5], Bergstra et al. [6, 7], Hutter et al. [25], Maclaurin et al. [42], Snoek et al. [58] and Swersky et al. [60]. For a deeper discussion of *hyperparameter tuning* literature, refer to Appendix C.1.

- **Alternative Models Exploration, Testing, and Validation (ATV):** A prediction taskcan generally be solved using multiple ML techniques. Currently, automation work related to the selection, validation and finalization of models is broadly categorized under the subfield *AutoML*. We are instead referring to this step as ATV, as we view *AutoML* to be an intelligent solution with the far broader goal of automating an end-to-end machine learning process. Perhaps the past decade's most widely studied area,[1] major efforts in this direction include Feurer et al. [18], Swearingen et al. [59], and Thornton et al. [61]. A number of commercial systems already exist for this task. Below we briefly describe the two subtasks of ATV, and point the reader to some automation systems, open source libraries and commercial systems.
  - **Alternative Models Exploration:** Data scientists often spend time exploring alternative models and deducing their pros and cons. A recent DARPA-led initiative, **"Data-Driven Discovery of Models" (D3M)**, focuses on alternative strategies for model exploration, or the automation of "complete" data mining (Lippmann et al. [37]). It has inspired several other tools, such as AlphaD3M, an AutoML system based on edit operations performed over machine learning pipeline primitives (Drori et al. [14]), and ML Bazaar, a composable framework for developing what are called "end-to-end" ML systems (Smith et al. [57]). In a similar vein, Olson et al. [47] developed a **tree-based pipeline optimization tool (TPOT)** that automatically designs and optimizes machine learning pipelines for a given domain, while Cashman et al. [8] studied how visual analytics can play a valuable role in the automated model discovery process. Following the growing interest in deep neural networks, multiple researchers including Baker et al. [2], Liu et al. [38, 40], Pham et al. [50], Real et al. [52], Zoph and Le [71], and Zoph et al. [72] have proposed neural or reinforcement learning models to automatically search optimal neural network structures. For a more detailed review of existing literature on alternative models exploration, refer to Appendix C.2.
  - **Evaluation:** After selecting the best-performing model for the training set, the data scientist must evaluate this model on the testing set and report the statistical significance of these evaluations. Because this step is so important, the machine learning community has put significant effort into comparative evaluation of different machine learning models. Some recent efforts include benchmarking of deep learning software tools by Shi et al. [55], hyperparameter auto-tuning methods by Gustafson [21], deep reinforcement learning approaches by Duan et al. [15] and bio-medical data-mining methods by Olson et al. [48]. More details about evaluation and benchmarking can be found in Appendix D.

- **Result Summary and Recommendation (RSR):** The final and arguably most important job of a data scientist is to summarize the findings and recommend the most useful tasks to domain experts. For each task, recommendations can be made at the level of models, features, or computational overhead. This step is unsystematic and done mostly manually.

[1] A quick search of "AutoML" in Google Scholar will bring up thousands of papers, with the top results cited over 100 times.

The different libraries and tools discussed so far typically make up one component within a larger system that aims to manage several practical aspects of machine learning, such as parallel and distributed training, tuning, model storage, and even serving and deployment. Such large systems include **Auto-Tuned Models (ATM)** [20, 59, 65], as well as commercial systems like Google AutoML[2] and Amazon Forecast.[3] For more details about these complex machine learning systems, refer to Appendix E.

Table 1 provides a comprehensive summary of current tools that aim to automate particular subtasks (or sets of subtasks) of the machine learning process, with more details in the appendix. It is organized by subtask, and lists the notable research contributions, survey articles, open source systems and commercial systems that relate to that subtask.

3 THE CURRENT FLOW OF END-TO-END MACHINE LEARNING PIPELINES

The primary goal of *AutoML* is to reduce the manual effort involved with machine learning technologies, thus accelerating their deployment. Consequently, various systems have attempted to minimize the work required to perform certain steps of the machine learning development workflow (see Table 1 and the appendix for a detailed discussion). For example, DeepDive/Snorkel [51] is a general, high-level workflow support system that helps users label and manage training data and provides high-level support for model selection. As previously mentioned, however, developing ML solutions still involves a lot of manual work. To design a truly automated system, it is important to address the bottlenecks in the current process. To better visualize these bottlenecks, we present in Figure 2 a flowchart showing the end-to-end machine learning process. For each step in the flow, we outline the role of domain experts, the amount of manual work performed by the data scientist, and the communication required between the two.

Figure 2 highlights key subtasks that need more attention from the research community if full automation is to be achieved. Models and software developed so far have enabled or made significant progress toward the automation of DCC, ML, FE and ATV. The biggest communication bottlenecks currently happen in two places: TF and RSR, both of which require a significant amount of manual work. Prediction engineering also can lead to prolonged back-and-forth between the data scientist and the domain expert.

These tasks are still largely unstructured and are performed manually through trial and error. The community has been reluctant to focus on them, possibly due to their human-centric nature—these tasks involve significant human interaction, and their evaluation is also largely subjective. For example, without a robust user preference model, it is hard to identify "interesting" prediction tasks and eventually make useful automatic recommendations. A lack of systematic studies of these problems prevent the completion of a fully automated pipeline that can transform a dataset into a useful predictive model and use it to mine insights from it.

It should be noted that existing AutoML solutions have primarily approached these tasks from a software systems perspective without much theoretical analysis. Another important thing to note is that domain experts currently have minimal access to the FE, ML, and ATV sub-tasks, and as a result are left out of these parts of the process—instead waiting for the data scientist to figure it out and get back to them, perhaps with some exciting results. Finally, once business leaders agree to deploy a particular predictive model, deployment, maintenance and monitoring are also handled by data scientists and software engineers.

[2] https://cloud.google.com/automl/.
[3] https://aws.amazon.com/blogs/aws/amazon-forecast-time-series-forecasting-made-easy/.

Table 1. A Summary of Contributions in the Broader Field of AutoML

Task	Research Contributions	Survey Articles	Open Source Systems	Commercial Systems
Data Visualization,	Miao et al. [44]	Ilyas et al. [27]	Open Refine [U1]	Trifacta Wrangler [U2]
Cleaning and Curation	Zhang et al. [69]	Chu et al. [11]	Drake [U3]	TIBCO Clarity [U4]
	Minnich et al. [45]		Open Source Data Quality and Profiling [U5]	Winpure [U6]
	Cheng et al. [10]		Talented Data Quality [U7]	Data Ladder [U8]
	Krishnan et al. [36]			Data Cleaner [U9]
	Chu et al. [12]			Cloudingo [U10]
	Haas et al. [22]			Refier [U11]
				IBM Infosphere Quality Stage [U12]
Feature Engineering	Katz et al. [32]	Wang et al. [64]	FeatureTools [U13]	FeatureTools [U13]
	Kanter and Veeramachaneni [30]	Chandrashekar and Sahin [9]	featexp [U14]	
	Mountantonakis and Tzitzikas [46]	JI et al. [28]	MLFeatureSelection [U15]	
	van den Bosch [62]	Sheikhpour et al. [54]	Feature Engineering & Feature Selection [U16]	
	Khurana et al. [34]	Liu et al. [39]	FeatureHub [U17]	
	Kaul et al. [33]		featuretoolsR [U18]	
			Feast [U19]	
			ExploreKit [U20]	
Hyperparameter	Bergstra and Bengio [5]	Vanschoren [63]	Hyperparameter Hunter [U21]	Amazon Sagemaker [U22]
Tuning	Snoek et al. [58]	Hutter et al. [26]	hyperband [U23]	BigML OptiML [U24]
	Hutter et al. [25]		Hyperboard [U25]	Google HyperTune [U26]
	Bergstra et al. [7]		SHERPA [U27]	Indie Solver [U28]
	Bengio [3]		Milano [U29]	Mind Foundry OPTaaS [U30]
	Bergstra et al. [6]		BBopt [U31]	sigopt [U32]
	Swersky et al. [60]		adatune [U33]	
	Maclaurin et al. [42]		gentun [U34]	
			optuna [U35]	
			test-tube [U36]	
			Advisor [U37]	
Alternative Models	Thornton et al. [61]	He et al. [23]	AutoKeras [U47]	H2O Driverless AI [U48]
Exploration	Feurer et al. [18]	Elsken et al. [17]	AdaNet [U49]	Cloud AutoML [U50]
	Swearingen et al. [59]	Zöller and Huber [70]	PocketFlow [U51]	C3 AI Suite [U52]
	Lippmann et al. [37]		automl-gs [U53]	FireFly [U54]
	Zoph and Le [71]		MLBox [U55]	Builton [U56]
	Zoph et al. [72]		Morph-net [U57]	Determined AI Platform [U58]
	Liu et al. [40]		ATM [U59]	
	Liu et al. [38]		TransmogrifAI [U60]	
	Real et al. [52]		RemixAutoML [U61]	
	Pham et al. [50]			
	Baker et al. [2]			
Evaluation and Benchmarking	Shi et al. [55]	Gijsbers et al. [19]	OpenML100 [U62]	
	Gustafson [21]	Klein et al. [35]	OpenML-CC18 [U63]	
	Duan et al. [15]		AutoML Challenges [U64]	
	Olson et al. [48]			
	Eggensperger et al. [16]			

(Continued)

Table 1. Continued

Task	Research Contributions	Survey Articles	Open Source Systems	Commercial Systems
Systems	Swearingen et al. [59] Golovin et al. [20] Wang et al. [65]		Scikit-learn TPOT [U39] [47] Auto-Sklearn [U41] AlphaD3M [14] ML Bazaar [57] PyTorch	DotData [U38] IBM AutoAI [U40] Azure Machine Learning [U42] Google Could AI platform [U43] Amazoon AWS service [U44] Data Robot [U45] Amazon Forecast [U46] RapidMiner [24] Orange [13]

For each subtask, we list related and notable research contributions, survey articles, open source systems, and commercial systems. For more details, refer to the appendix.

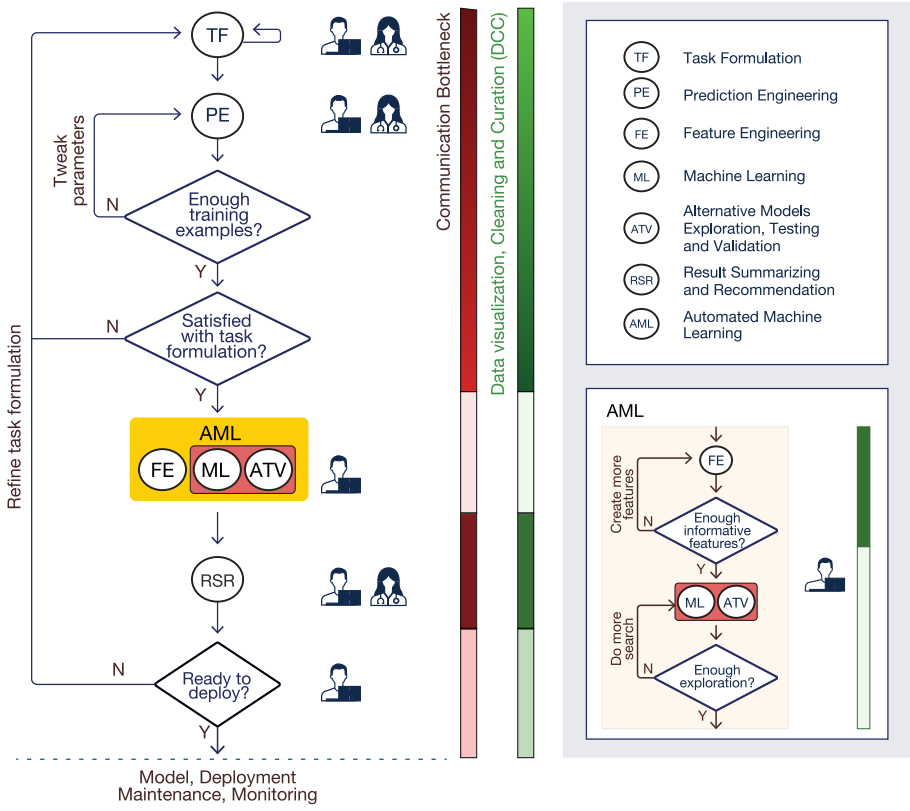

Fig. 2. A flowchart showing the machine learning process. This chart highlights points of interaction between domain experts and data scientists, along with bottlenecks.

4 CLASSIFYING AUTOML SYSTEMS BASED ON LEVELS OF AUTOMATION

To better characterize the current drive toward automation by the ML community, we first define a set of taxonomic levels to represent different degrees of automation provided by systems developed so far. These levels are designed such that lower levels mean less automation and more

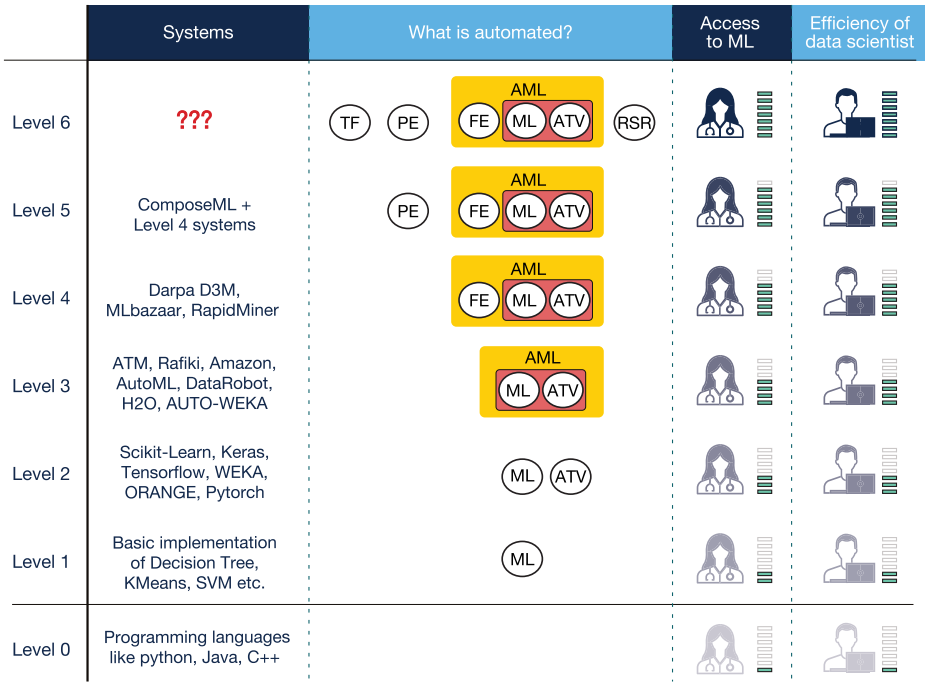

Fig. 3. Levels of automation possible for end-to-end machine learning endeavors. Level 0, the lowest level, is entirely manual and requires people to develop and write software for the whole process. Level 6, the highest level, is entirely automated from task formulation all the way to result summary and recommendation. This figure provides examples of existing systems at each level, along with a color gradient indicating accessibility to the domain expert. As is shown here, increased automation has two simultaneous effects. First, it enables domain experts to make use of machine learning. Second, it increases the productivity of data scientists, because their manual work declines as the levels increase.

manual work, and higher levels mean more automation and less manual work. A system's level also correlates with a domain expert's ability to interact with that system despite a lack of ML expertise. Figure 3 illustrates the proposed taxonomic levels, which we define below. These levels have no absolute meaning but are rather a way to organize machine learning solutions based on the degree of automation they provide.

4.1 *Level 0: No Automation*

All ML tasks are performed manually by hand-coded implementations. No automation is supported.

Real-world Example: Machine learning researchers, who study and develop new machine learning models, mostly work at this level as they build core machine learning algorithms from scratch.

4.2 *Level 1: Only ML Automated*

These tools provide basic implementations of core machine learning algorithms. This is the minimal level of automation supported by a system. Tools at this level are generally developed by machine learning researchers and used by data scientists.

Real-world Example: C++ implementation of Support Vector Machine Classifier, which can be used as a library by data scientists. However, training set construction, label creation, feature construction, hyperparameter tuning, and so on must be carried out manually by the data scientist.

4.3 *Level 2: ML + ATV Automated Separately*

While these automated tools are popular with data scientists, they require a lot of manual work and are not at all accessible to domain experts.

Real-world Example: When a training set and well-defined feature sets are already available, data scientists can use a Level 2 tool like Weka to perform the actual learning and hyperparameter tuning tasks.

4.4 *Level 3: ML + ATV Automated Jointly*

Some automated tools are provided at this level, while the rest of the work is done manually by the data scientist. Domain experts still do not have direct access to tools at this level.

Real-world Example: Level 3 tools like AutoWeka allow users to perform learning and hyperparameter tuning jointly, and thus require less manual work than Level 2 tools.

4.5 *Level 4: ML + ATV + FE Automated*

A domain expert can interact minimally with a system at this level. However, the data scientist must still perform a significant amount of manual work.

Real-world Example: Consider a wind energy company that owns a number of turbines, each equipped with many sensors. Predicting turbine failure in advance can save the company headaches and reduce their costs. However, numbers generated by the sensors are only meaningful to wind energy experts. A data scientist working on this problem must spend long hours with these experts, to understand the sensor data and what indicates turbine failure so that s/he can create meaningful features. Level 4 tools like MLBazaar aid the data scientist by automating this feature construction step. They can also actively engage domain experts by asking them to verify the quality of automatically created features.

4.6 *Level 5: ML + ATV + FE + PE Automated*

Domain experts can comfortably interact with systems at this level, and minimal manual work is required from the data scientist.

Real-world Open Problem and Potential Application: Imagine a railway company that continuously monitors signals from different engine sensors, and wants to predict future failure of engine components. A data scientist working on this problem must collaborate with domain experts, including engineers, programmers, and even conductors, to fully understand how failure is defined and create effective training examples for learning the predictive model. A Level 5 AutoML tool will be very useful for everyone in this scenario, because it can help users automatically create a good-quality training set for supervised learning, as well as a validation set for testing purposes.

Automatic prediction engineering distinguishes a Level 5 AutoML solution from those at lower levels. There are many open problems associated with automatic *prediction engineering*, because it requires identifying missing information from natural language problem descriptions and filling them with of computable metrics. For example, consider the problem of forecasting future engine failures due to inclement weather. Two fundamental challenges here involve: (1) Computationally defining "inclement" weather and (2) Automatically collecting the "right" training examples from historic data. Thus, automatic prediction engineering is essentially a hybrid human-computer interaction and data science problem, and will require joint investigation from both of these communities.

4.7 *Level 6: ML + ATV + FE + PE + TF + RSR Automated*

Level 6 provides the maximum amount of automation and requires the least amount of manual work. Domain experts can easily use Level 6 systems by themselves.

Real-world Open Problem and Potential Application: Imagine a hospital that collects patient **electroencephalography (EEG)** results on a regular basis, and wants to use them to predict which patients are at higher risk for epilepsy. Doctors are not experts at building such predictive models; however, they do have enough medical knowledge to understand EEG log data and define prediction tasks rigorously. Level 6 AutoML tools allow doctors and stakeholders to directly input their goals, and can formulate relevant prediction problems almost autonomously and recommend relevant prediction tasks to users along with the corresponding results.

The two key functionalities distinguishing Level 6 automation tools from those on lower levels are: (1) *Automatic Prediction Task Formulation* and (2) *Recommendation*. Both functionalities come with numerous open questions that warrant rigorous academic investigation. For example, one way to formulate *automatic prediction task formulation* is to frame it as an information extraction task within a dialog-based framework. In this case, challenges include but are not limited to: How can we develop a general representation for a machine learning task? How can we extract task-specific parameters through active dialog with users? How can we efficiently navigate a very high-dimensional search space containing all possible prediction tasks?

Although recommendation is more of a classic *information retrieval* problem, the prospect of automating it invites the following questions: How can we evaluate the quality of a prediction task itself? How can we revise a recommendation model by leveraging active user feedback? How can we capture business goals while recommending prediction tasks? And so on.

The AutoML ecosystem is sprawling. Achieving full automation will require expertise from many subfields. Unfortunately, most existing AutoML solutions focus almost entirely on hyper-parameter tuning and feature engineering, and have primarily approached the problem from a software engineering and integration perspective. In developing our taxonomy, we aim to highlight core theoretical problems associated with different levels of machine learning automation, and invite people to recognize and investigate them systematically.

Using Our Tiered Taxonomy to Classify AutoML Tools.

Classifying a machine learning tool into one of these seven levels simply requires identifying which data science subtasks (as presented in Section 2) it has automated. The subtasks themselves act as features for classifying the AutoML tools, where each feature assumes a binary value: "yes" means automation is provided, while "no" means it is not. For example, MLbazaar [57] provides automation for ML, ATV and FE but does not provide automation for TF, PE, and RSR. Thus, MLbazaar is categorized as an AutoML system with Level 4 automation. As another example, the popular data-mining tool WEKA [67] provides automation for ML and ATV separately, and thus will be considered an AutoML system with Level 2 automation. An extension of WEKA, i.e., AUTO-WEKA [61], provides automation for ML and ATV separately and thus will be categorized as a Level 3 tool.

5 TOWARD LEVEL 6 AUTONOMY FOR AUTOML

As Figure 3 clearly demonstrates, automation efforts so far have primarily been limited to tools that belong in Levels 1 through 4. The field is sorely lacking in both Level 5 and Level 6 automation tools—the only ones that allow domain experts to contribute heavily and take the load off of data scientists.

How can we make the machine learning process more efficient and automatic, and relieve the communication bottleneck between data scientists and domain experts? One option is to enable

domain experts to directly engage with big data without requiring them to have detailed knowledge of machine learning. Although this is a highly ambitious goal, if reached, it will expedite the implementation of machine learning systems and allow a far broader audience to make use of ML. Imagine an intelligent agent that can guide domain experts in big data exploration and analytical tasks, taking care of certain subtasks automatically and thus filling the role usually played by a *human* data scientist. This kind of system would make big data analytics more appealing and accessible to business entities, and increase the productivity of human data scientists significantly by automating a large portion of their manual labor. Only a Level 6 AutoML tool could provide such support.

The importance of TF (a subtask that is not currently automated) and PE (one that rarely is) have been substantiated by leading practitioners in the software industry. In popular tutorial materials, practitioners discuss the value of TF and PE, and their high impact on the use and value of the overall ML model [1]. For example, someone seeking to predict user enjoyment of a mobile app—which is difficult to define and model—can get a similar answer by focusing on a simpler, easier-to-define metric like "number of user sessions per week." According to these practitioners, defining the right prediction goal is often more important than choosing a particular algorithm, and a few hours spent at this stage in the process can save many weeks of work downstream.

Previous work has laid some foundations for automating *prediction engineering*. Schreck and Veeramachaneni [53] designed a formal language, called *Trane*, to describe prediction problems over relational datasets, allowing data scientists to specify problems in that language. Another related work, MLFriend [68], is an interactive framework that defines useful prediction problems on event-driven time-series data, and is designed to appeal to domain experts. In addition, Microsoft has introduced an alternative training paradigm, *Machine Teaching*, which actively teaches domain experts to train machine learning models [56]. The authors have proposed a set of principles for teaching those new to the machine learning field, including a universal teaching language, "realizability" of all target concepts, a rich and diverse sampling set, and distribution robustness. The authors posit that bringing new people on board through active teaching will accelerate innovation and empower millions of new uses for machine learning models. While these are steps toward the goal of an end-to-end automated machine learning pipeline, they are limited by their focus on a particular data science function (e.g., Machine Teaching focuses on the training process, while MLFriend and Trane focus on defining the prediction task) or a particular type of data (e.g., MLFriend is designed for event-driven time-series data), as well as by the absence of an user-centric prediction task recommender system, which all lack.

How can we build an end-to-end, automated machine learning pipeline—in our conception, an intelligent data science agent with a autonomy level of 6? In the following sections, we provide a roadmap for future AutoML research that could lead to the development of such a tool, along with important challenges and opportunities that will likely come with achieving this ambitious goal.

6 CHALLENGES AND OPPORTUNITIES

This section presents the unique challenges of developing a level 6 AutoML agent in terms of design and learning goals. We envision that fully solving these challenges will require more research, and so present some motivating examples to inspire the community to jointly pursue them.

Let's Do a Case Study: As most real-world data come in the form of event-driven time series, and the primary goal of a Level 6 AutoML agent is to enable domain experts to perform predictive analysis directly on such data, we focus on event-driven time-series data in this case study. We

Table 2. Flight Delay Dataset details

Attribute Type	Attribute Name and Meanings
Timestamp	Date, Day_of_week scheduled_departure_hour scheduled_time, elapsed_time (the scheduled trip time and the actual time)
Entities	Airline, Flight_number, Tail_number Origin_airport, Destination_airport
Categorical Attributes	cancelled_status, Cancellation_reason (whether the flight is canceled and reason for cancellation)
Numerical Attributes	Departure_delay, Arrival_delay (the departure delay and arrival delay in minute) Air_system_delay, Security_delay, Airline_delay, Late_aircraft_delay, Weather_delay (delay caused by different reasons in minute)

picked a dataset we call **Flight Delay**,[4] which contains flight records for the United States from 2015. As this dataset has a timestamp associated with each event, we pick the time attribute, entity attributes, numerical attributes, and categorical attributes, which we use to generate prediction tasks. For simplicity, we remove unsupported attributes like sets or natural language. Table 2 shows the attributes we used to generate possible prediction tasks. Using this dataset, we conducted an exploratory case study to understand how a Level 6 AutoML agent could be used in a real-life scenario.

6.1 Challenges in Task Formulation

Our key observation is that across domains, a prediction task is usually characterized by three components: An *outcome function*, the *prediction window* over which this outcome is calculated (denoted `window`), and the *prediction horizon* over which one wishes to predict this outcome (denoted `lead`). Abstracting the `prediction window`, `lead` and other parameters away from the outcome function allows us to focus on the most important point of translation from human intuition to problem specification, i.e., the definition of the outcome function itself. Human data scientists often translate an abstract goal like "understanding customer engagement" into a computable quantity that can be predicted. For example, engagement with a particular website could be quantified as *the number of times a customer visits the website during a week*. Here, the outcome is *the number of visits to the website* and the `prediction window` is one week. Prediction task formulation is one of the most critical steps of machine learning; even industry leaders at Facebook have specifically emphasized that: "*the right set-up is often more important than the choice of algorithm*" [1]. What this means for a Level 6 AutoML agent is that we need a formal language to represent prediction tasks accurately, and that this language must be general enough to translate an abstract goal into a computable quantity.

(1) **Prediction Task Expression Language:** To develop a Level 6 AutoML agent, it is necessary to have a language that can be used to specify prediction tasks. We will call this **Prediction Task Expression Language (PeTEL)**. We argue that a complete PeTEL needs to successfully express two basic components: (1) a generic *Outcome Function* and (2) *Search Parameters* (`prediction window`, `lead`, and others) to materialize that outcome function. Previous work, including Trane [53] and MLFriend [68], has made progress on a reasonable design for Search Parameters, as well as simple expressions for outcome function. However,

[4] https://www.kaggle.com/usdot/flight-delays.

Table 3. Operators

Operation Set	Supported Ops	Supported Types
Filter	`all_fil`	None
$\mathbf{O}_f$	`greater_fil`, `less_fil`	Numerical
	`eq_fil`, `neq_fil`	Categorical/Entity
Aggregation	`count_agg`	None
$\mathbf{O}_g$	`sum_agg`, `avg_agg`,	Numerical
	`min_agg`, `max_agg`	Numerical
	`majority_agg`	Categorical/Entity

expressing an outcome function still remains a significant challenge; specifically, how can we express an outcome function that is easily understandable by domain experts with little or no experience in machine learning, so that they can play with it? Fulfilling this goal will require a much larger effort.

As with any language, PeTEL should provide constructs for variables, operators and functions. What should these look like? Variable types can be as simple as integers, real-valued numbers, and categorical attributes, or they can be more complicated variables, like entities and events. The details of these types will vary based on implementation, while the operators will depend on specific variable types. Some basic ones might include filter operators (equal to, greater than, etc.), aggregation operators (count, sum, average, etc.), and smoothing operators (to handle missing values in the data). Table 3 shows some examples of operators for a Level 6 AutoML agent, along with their supported types. For instance, take the following prediction goal:

For each airline, predict the maximum delay suffered by any of its flights.

This prediction goal can be expressed by the following PeTEL expression:

```
Entity: AIRLINE
Filter: NONE
Aggregator: max_agg(<ARRIVAL_DELAY>)
```

(2) **Interpretability of a Task:** While PeTEL is about encoding an outcome function into a standard representation, the *interpretability* of a task is essentially the other side of the coin. That is, how can real humans decode/comprehend the actual target outcome from that standard language expression? For example, consider the following PeTEL expression: While PeTEL is about encoding an outcome function into a standard representation, the *interpretability* of a task is essentially the other side of the coin: How can humans translate the standard language expression into an actual target outcome? For instance, consider the following PeTEL expression:

```
Entity: AIRLINE
Filter: eq_filter(<CANCELLATION_REASON, 'bad_weather'>)
Aggregator: majority_agg(<DESTINATION_AIRPORT>)
```

Although people with basic knowledge of structured query language may understand the meaning behind this expression, others will likely have more trouble. However, if a Level 6 AutoML agent translates the PeTEL expression into the following natural language form, then it becomes much clearer to the user:

For each airline, predict which destination airport will have the largest number of flights canceled tomorrow due to inclement weather.

Thus, translating PeTEL expressions into human-readable language is a core challenge that must be addressed to create a Level 6 AutoML agent.

6.2 Challenges in Prediction Engineering for Identifying "Promising" Tasks

Because the space of possible prediction tasks grows exponentially with the number of attributes (in the dataset) as well as the number of operators (supported by PeTEL); executing full AutoML pipelines for all these tasks is not a computationally feasible option for an interactive Level 6 AutoML agent. One way to address this issue is to filter out uninteresting and invalid tasks, and create a priority list of *promising* tasks that can be evaluated with a feasible amount of computational effort. The primary challenge of identifying *promising* tasks lies in how we define and quantify the *promise* of a prediction task. This is a philosophical question with no single right answer. To simplify things, we identify four metrics that we believe are indicative to some degree of the promise of a prediction task, and discuss the challenges and design choices involved with their computation.

(1) **Task Validity:** While PeTEL can generate many different prediction tasks by combining operators and variables, not all combinations will yield a valid task; indeed, many will be invalid. For example, one prediction task for the flight delay dataset can be expressed by the following PeTEL expression:

```
Entity: <AIRLINE>
Filter: less_filter(<ARRIVAL_TIME>, <DEPARTURE_TIME>)
Aggregator: count_agg(None)
```

where a filter is applied first to check that `ARRIVAL_TIME < DEPARTURE_TIME`, and then a count operation is performed over the filtered records after grouping them by airline. Although this is a syntactically correct task, it is not semantically meaningful. Indeed, the natural language translation of the PeTEL expression would be the following:

For each airline, predict how many flights tomorrow will arrive at the destination before they depart.

This is clearly an invalid task, as it does not make any sense. Thus, one very important challenge for a Level 6 AutoML agent is to figure out how to filter out invalid tasks.

(2) **User Preference:** Understanding which sorts of prediction task tend to be preferred by a particular user is crucial for effectively identifying *promising* prediction tasks. Capturing *User Preference* has two major benefits: (1) we can recommend highly relevant task to users and (2) we can easily filter out tasks the user dislikes, saving time and computational resources. However, modeling *User Preference* in this context is tricky, and often specific to a particular user and their domain. For example, an analyst at an airport authority may be more interested in predicting whether their airport will have an unusual number of delayed flights so that they can accommodate passengers, while an analyst at an airline may be more interested in whether their own flights will be delayed. Although both tasks are valid, they may not be equally interesting to different users. Thus, the effectiveness of filtering *promising* tasks for a user depends heavily on accurate modeling of that user's preferences. In the absence of a pre-trained user model, an active learning method for capturing user preferences, such as the *Machine Teaching* perspective introduced by Microsoft, can become really handy [56].

(3) **Potential Business Value:** Another metric for filtering out less *promising* tasks involves quantifying their potential business value and discarding tasks with low scores. Assuming that a particular prediction task can achieve reasonably high accuracy, would it impact the business significantly? If not, then it may not be worth the effort to train a machine learning

model to carry it out. *Potential Business Value* is often very hard to quantify, and the best option may be to ask domain experts with long-term experience to foresee the impacts of such models.

(4) **Sufficient Training Examples:** Given a valid and highly interesting prediction task, another important metric related to *promise* is the number of training examples it's possible to extract from the available data. If this number is very low, then it may not be very meaningful to train a model on such a small corpus. In the case of classification tasks, one must also ensure that enough training examples from each class are present in the training data.

One can compute the overall promise of a prediction task by combining these four metrics: User Preference, Task Validity, Business Potential Value, and Sufficient Training Examples. Again, this is not an absolute standard by any means, and this remains an open problem.

6.3 Challenges in Result Summary and Recommendation

As mentioned in Section 2, RSR is a data scientist's most important job, and is still done largely manually, without any systematic structure. To realize a Level 6 AutoML agent, even this must be done automatically. The goal here is to rank prediction tasks based on their utility scores, and to present the top-k prediction tasks to domain experts or other users. Below we present a brief summary of major challenges associated with the automation of the *RSR* sub-task.

6.3.1 Evaluation Challenges. Prediction task recommendation essentially boils down to ranking tasks based on their utility to the user. But how should we evaluate the utility of a prediction task, and what does "utility" really mean in this context? In other words, how do we evaluate the performance of a prediction task recommender system? We hypothesize that utility scores can be computed as a function of the task-specific metrics discussed in Section 6.2 including pre-AutoML metrics like *User Preference*, *Task Validity*, and so on, as well as post-AutoML metrics like *Prediction Accuracy*, *Computation Time*, and so on. Some of these metrics can be computed automatically, like *Sufficient Training Examples* and *Prediction Accuracy*, while some metrics, like *User Preference* and *Potential Business Value*, are highly dependent on actual users—in this case domain experts or business leaders. Thus, utility appears to be a complex metric composed of simpler metrics, some of which are computed automatically and some of which are collected through human annotations and feedback. However, it is unclear which types of user feedback—user rating, pairwise comparison, partial ranking, and so on—are the most appropriate for gathering annotations.

6.3.2 Representation Challenges. The next challenge in designing a prediction task recommender system is finding a way to represent prediction tasks for the machine learning setup. This involves transforming the prediction task descriptions into a feature vector. While the pre-AutoML and post-AutoML metrics can certainly be used to compute meaningful numerical features, what about features related to task description? How do we convert the PeTEL expressions into a set of features suitable for machine learning? Should we choose a binary vector over the space of the attribute set, aggregation and filter operations? Should we also consider the natural language descriptions through some simple language models, like *unigram* or *bigram*? Or, given the promise of neural architectures, should we build a distributed representation or embedding for prediction task featurization? A detailed investigation of these questions will be required to reach a conclusion.

6.3.3 Learning Challenges. Given a particular *utility* metric and feature representation for prediction tasks, a crucial challenge is finding the best way to learn this *utility* metric. As with any learning system, this comes with basic challenges including the following: How do we create an effective training corpus? How should we model the ranking function of the recommender system? What is the ideal objective function in this case?

Table 4. Example Prediction Tasks on the *Flight Delay* Dataset

PeTEL: `Entity: AIRLINE, Filter: NONE, Aggregator: max_agg(<ARRIVAL_DELAY>)` **Intermediate:** For each <AIRLINE> predict the maximum <ARRIVAL_DELAY> in all related records **Natural:** Predict the delay time of each airline's most delayed flight tomorrow
PeTEL: `Entity: AIRLINE, Filter: less_filter(<ELAPSED_TIME>), Aggregator: sum_agg(<AIRLINE_DELAY>)` **Intermediate:** For each <AIRLINE> predict the total <AIRLINE_DELAY> in all related records with <ELAPSED_TIME> less than __ **Natural:** Predict the total flight delay tomorrow for each airline's short-haul flights
PeTEL: `Entity: AIRLINE, Filter: greater_filter(<AIRLINE_DELAY>), Aggregator: majority_agg(<DESTINATION_AIRPORT>)` **Intermediate:** For each <AIRLINE> predict the majority of <DESTINATION_AIRPORT> in all related records with <AIRLINE_DELAY> greater than __ **Natural:** For each airline, predict which destination is the most likely to have delays caused by the airline itself
PeTEL: `Entity: ORIGIN_AIRPORT, Filter: greater_filter(<DEPARTURE_DELAY>), Aggregator: majority_agg(NONE)` **Intermediate:** For each <ORIGIN_AIRPORT> predict the number of records with <DEPARTURE_DELAY> greater than __ **Natural:** Predict how many flights departing from this airport will be delayed more than 1 hour
PeTEL: `Entity: AIRLINE, Filter: eq_filter(<CANCELLATION_REASON>), Aggregator: majority_agg(<DESTINATION_AIRPORT>)` **Intermediate:** For each <AIRLINE> predict the majority of <DESTINATION_AIRPORT> in all related records with <CANCELLATION_REASON> equal to __ **Natural:** Predict which destination airport has the largest number of flights canceled tomorrow due to weather

"PeTEL" is the structured representation of the problem, including the entity, attributes and operations. "Intermediate" is an automatically generated natural language description of the problem. "Natural" is the human interpretation of the problem.

As a starting point, we identify four different ways to rank prediction tasks to provide users with recommendations. Below, we provide a brief summary of each.

(1) **Static Ranking:** The simplest way to recommend prediction tasks is to rank them according to their *utility* score, and present tasks with the highest scores. Through initial experiments conducted on the *Flight Delay* dataset, we present some examples of highly recommended tasks in Table 4, which we curated through manual inspection. This static ranking list can be presented with or without each tasks' *utility* score; however, it is currently unclear whether user experience will vary based on whether this *utility* score is shown or not.
Static Ranking is appropriate when ranking is based on well-defined numerical scores that can be computed automatically. For example, if we want the ranker to show tasks with high prediction accuracy only, then static ranking is sufficient for our purposes. Static Ranking may also work in cases where we have a very robust user model that has already been learned on a large amount of historical data. For example, if we know the preferences of a particular user very well, then Static Ranking may work for that user.

(2) **Interactive Ranking with Feedback:** This setup is required in cases when the *utility* function depends heavily on the user and a robust model for user preference is not available beforehand. In this setup, a Level 6 AutoML agent can first recommend a set of tasks that have been scored highly by the recommender system, and ask the user to label these tasks as "useful" and "not-useful." By looking at this feedback, a Level 6 AutoML agent can actively learn the user's preferences and revise the recommendation model to accommodate them (see Ref. [41] for details on how active learning can be used for ranking tasks). This recommendation-modification loop can continue until the user is satisfied.

(3) **Recommendations from Previous Experiences:** This setup is useful as a starting point in cases when the problem domain is not known historically. For example, a recommender system may have been previously trained on car rental data, but not on truck rental data;

these two rental systems may share similarities and differences the specifics of which are unknown. One reasonable way to approach this problem is to transfer knowledge from the car rental domain and use it for initial recommendations in the truck rental domain, an increasingly popular strategy known as transfer learning [31, 43].

(4) **Combining Interactive Ranking with Transfer Learning:** A more sophisticated way of recommending would be to start with a ranker that's been learned on a similar domain with sufficient training examples, and interactively update that ranker by getting feedback from users of the target domain. Going back to the car vs. truck rental example, one can start with the recommender model learned from the car rental domain to provide recommendations in the truck rental domain while gathering feedback from users about the quality of the recommendations. Once the system gathers more information about the truck rental domain, the recommender model can be revised to capture this new information and better serve the target domain.

6.3.4 Diversity Challenges. Another challenge for the recommender system is to ensure that the list of recommendations includes a diverse set of tasks. For example, consider the following prediction tasks:

Task 1: predict the maximum *weather_delay*
Task 2: predict the average *weather_delay*
Task 3: predict the average *security_delay*
Task 4: predict the total *security_delay*

If we were to recommend two prediction tasks from this set of four, then which two would ensure more diversity in terms of attributes and operations? Task 1 and 2 predict *weather_delay*, while, Task 3 and 4 predict *security_delay*. Clearly, task-set $\{1, 3\}$ is more diverse than task-set $\{1, 2\}$, while, task-set $\{2, 3\}$ is less diverse than task-set $\{2, 4\}$. However, the degree of diversity required depends on the user's preference, and there is not yet any clear, general standard for evaluating it.

7 HUMAN-AI INTERACTION IN LEVEL 6 AUTOML AGENT

Ultimately, human analysts and decision-makers must be convinced that prediction tasks recommended by the Level 6 AutoML agent are indeed useful and interesting. This goal brings up several interesting challenges and opportunities in Human-AI interaction, some of which we highlight below.

- **Problem Space Understanding:** Human analysts need to have some understanding of the problem space—the set of all prediction tasks that could possibly be generated by the Level 6 AutoML agent—to grasp the context in which the data science process takes place. This closely relates to the problems of looking at raw data and identifying useful tables and attributes, understanding the frequency at which the data are collected and made available, and so on. The domain expert should recognize that the Level 6 AutoML agent can enumerate only a subset of this infinitely large problem space.
- **Prediction Task Understanding:** When the Level 6 AutoML agent presents a prediction task to a human analyst, what form should this presentation take? Multiple interfaces are possible: a raw PeTEL expression, a basic generated natural language description, a more sophisticated natural language explanation, a visual representation of the task in terms of inputs, a visualization of prediction instances associated with that task, or something else entirely. Such decisions are quite important, as true understanding and accurate evaluation of a given prediction task may depend on how it is presented to the user.
- **Prediction Task Evaluation:** As described in Sections 6.2 and 6.3, there are many dimensions along which a prediction task can be evaluated. Human evaluators must be able to

judge performance along each of these dimensions, often making a difficult connection between the prediction task at hand and other processes or outputs within their organization.

- **Prediction Task Operationalization:** After the selection of prediction tasks, the job is in some ways just beginning. End-to-end machine learning pipelines must be developed and deployed to solve these tasks. This involves additional challenges, such as setting any free parameters for the task, or adjusting prediction windows. These may involve additional rounds of communication between the human analyst and the Level 6 AutoML agent—raising similar questions to those delineated above, as well as new ones.

8 THE LONG ROAD AHEAD

The AutoML community has provided us with powerful tools for executing machine learning tasks and producing predictive models. However, defining prediction problems themselves and navigating the enormous space of alternatives still poses significant challenges even for data scientists—let alone for domain experts and business leaders, who still struggle to directly access the power of big data analytics. Indeed, a significant portion of the machine learning process today requires involvement from a data scientist, making the whole process inefficient and inaccessible to a wider audience. In this review article, we explained the gravity of this problem by introducing a new taxonomy for AutoML solutions consisting of seven distinct autonomy levels, and designed to highlight gaps and limitations in current machine learning practice. We then presented our vision for a more intelligent agent (Level 6) to address this issue.

A Level 6 agent would provide two main benefits: (1) It would increase the productivity of data scientists by automating prediction task formulations and recommendations, and (2) it would enable domain experts to directly access and interact with the fascinating world of predictive machine learning, opening up new opportunities for them to apply this powerful tool to real-world problems. With this in mind, we make the following key observations. First, formulating a prediction problem is challenging, and there is currently no established standard for accomplishing this task systematically. Second, the key technical challenge of realizing a Level 6 AutoML agent boils down to designing an interactive prediction task recommender system capable of actively engaging the user.

We presented an exploratory case study using a real-life dataset to demonstrate the feasibility of actually realizing the currently hypothetical Level 6 AutoML agent. Our initial exploration reveals the many design and technical challenges that must be addressed to reach the eventual goal of automated data science—in particular automated task formulation, effective prediction engineering and the recommendation of useful tasks. Another crucial but orthogonal issue is the evaluation of machine learning models, as fair evaluation is critical to the success of AutoML. Indeed, the fair evaluation of ML models depends on accurate design choices made throughout the entire *data science* process: training and testing datasets must be created in such a way as to ensure an accurate and robust model is learned; benchmarking datasets and evaluation test beds must be built to precisely and reliably assess the model's accuracy and performance; training labels should be created in a way that ensures minimal annotation bias in the ML models; and so on. In summary, it is very important that a Level 6 AutoML tool ensures an unbiased and representative model evaluation, as this is critical to the success of AutoML in general.

We view a Level 6 AutoML agent as a tool that will increase the productivity of data scientists and domain experts. With this article, we aim to draw the attention of the broader data science and human–computer interaction community, and urge them to invest significant research efforts in this direction. The long road ahead demands an exciting interdisciplinary research approach involving multiple areas of computer science, especially human–computer interactions, information retrieval, databases, programming language design, and software engineering.

APPENDICES

A AUTOMATION IN DATA VISUALIZATION, CLEANING AND CURATION

A.1 Research Contributions

Framework-based Approach: One school of researchers primarily adopted a framework-based approach for data visualization, cleaning, and curation. For example, Miao et al. [44] introduced a data quality assessment and cleaning framework specifically for electronic health records. They used this framework to conduct a case study to estimate the risk factors for hip fracture patients who might need to be readmitted within one month of their surgery. During this case study, they identified 23 critical data quality problems, 16 of which were addressed by the proposed framework. In a similar effort, Zhang et al. [69] proposed a new framework called **Iterative Minimum Repairing (IMR)** to iteratively repair anomalies in time-series data. In each iteration, IMR tries to identify the most confident anomaly and repair it; thus, it minimally changes one point at a time. The authors have also designed an incremental algorithm for parameter estimation, which can reduce the complexity of parameter estimation from $O(n)$ to $O(1)$. To enhance the accuracy of repairs in later iterations, the authors tried to learn the temporal nature of errors in anomaly detection. The article also demonstrated that IMR method's performance is significantly better than other state-of-the-art approaches, including simple anomaly detection and constraint-based repairing.

Category-based Approach: Minnich et al. [45] introduced a cleaning process (called "ClearView") for online review mining purposes by categorizing noise into three categories. First is the syntactic noise, usually misspelled words, on which they perform two types of cleaning: character-level and word-level. The second is semantic noise, for which they analyze the structure of a sentence. More specifically, they used Stanford CoreNLP Parser and utilized the confidence score from the parser as a semantic correctness measure of a sentence. The third is rating noise, which refers to the discrepancy between valid text sentiment and star rating. They used an ensemble of sentiment classifiers to iteratively label the sentiment of a review to maximize the agreement with the star-rating provided by the user.

Cheng et al. [10] took an indicator-based approach for data quality assessment and associated cleaning strategy. They introduced four indicators for data quality assessment in wireless sensor networks - amount of data, correctness, completeness, and time correlation index measure - and studied the outcomes of different orders of cleaning strategy. They also proposed a strategy to avoid redundant cleaning operations and reduce cleaning expenses while ensuring high data quality. This strategy starts with computing the volume indicator of the dataset and determine if the cleaning process is necessary at all. If it is necessary, then they execute the cleaning process via completeness, time-related, and correctness indicators.

Active Approach: Krishnan et al. [36] used an active cleaning strategy while preserving convergence guarantees in the context of statistical model training. They proposed a new technique called "ActiveClean," which supports convex loss models while prioritizing data with result alteration potential. Specifically, *ActiveClean* suggests a sample of data to clean based on the data's value to the model and the likelihood that it is dirty. *ActiveClean* interleaves cleaning and training process iteratively by starting with a dirty model as initialization and taking gradient steps (cleaning a sample of records) toward a global solution (a clean model). For the same amount of data cleaned, proposed optimizations in ActiveClean can improve model accuracy by up to 2.5×. In comparison with techniques such as *Active Learning* and *SampleClean*, which are not optimized for sparse low-budget setting, *ActiveClean* achieves model with high accuracy where it cleans far fewer records.

End-to-end Systems: End-to-end cleaning systems have been recently studied by some researchers. Chu et al. [12] presented an end-to-end data cleaning system named "KATARA" that

uses trustworthy **knowledge bases (KB)** and crowdsourcing for data cleaning. Given a table, KB and crowd, KATARA interprets the table semantics with respect to KB, identifies wrong data, and recommends top-k possible repairs. A graphical interface is provided to assist users in tuning the parameters. Given a selected table and KB, KATARA will cross-validate each -tuple in the table with KB. If a -tuple is inconsistent with KB, then an ambiguity is raised. The ambiguity is then presented to the crowd to get feedback, which can help KATARA repair the data more accurately.

Another end-to-end data cleaning workflow is "Wisteria" proposed by Haas et al. [22]. *Wisteria* allows users to specify data cleaning plans and allows iterative revision of the plan. On each iteration, *Wisteria* presents a cost-aware recommendation to improve the accuracy of the plan by swapping in a new cleaning operator.

A.2 Open Source and Commercial Systems

There are several open-source and commercial systems publicly available for performing data visualization, cleaning, and curation. Below we present brief highlights of these systems.

- **Open Refine** [U1] allows users to clean and transform their data through a browser-based interface. This is installed in a client's machine; thus clients do not need to share their data unless they are contributing to the software. This system cleans data based on users decision.
- **Trifacta Wrangler** [U2] is a free cloud service that organizes and structures data automatically as soon as it is imported into the system. The system then suggests transformation and aggregation. Within a managed cloud platform, a team can share their work, schedule a workflow, and connect more data.
- **Drake** [U3] is a text-based work-flow tool that automatically resolves data dependency and calculates command list and order. It has HDFS support for multiple input and outputs and it includes host features to bring sanity to chaotic data processing workflows.
- **TIBCO Clarity** [U4] is an on-demand data preparation tool. It detects data types and pattern that is later used for auto-metadata generation. It categorizes data using some predefined facets on text patterns. It also provides numeric distribution for variance identification within the data and a configurable fuzzy matching algorithm for duplicate detection.
- **Open Source Data Quality and Profiling** [U5] is an integrated high performing data management platform. It also has Hadoop support to move files from a Hadoop grid and create Hive tables.
- **Winpure** [U6] provides a data cleaning product that automatically determines data impurities and suggests solutions about different data quality problems. It offers users different data transformations and data preparation operations through various templates. It also allows user to load data from different types of data sources and save transformed data in various data types.
- **Talented Data Quality** [U7] analyzes raw data using match pairing component that uncovers and labels suspicious data. Using those data, they build a classification model that predicts matches for new data sources.
- **Data Ladder** [U8] provides a highly visual data cleaning application that automatically profiles data from the start. They use pattern recognition and fuzzy logic for duplicate reduction and cleaning unstructured data. They assure data quality by providing a firewall that prevents bad data from third party through APIs.
- **Data Cleaner** [U9] includes three different types of operations: analyzers, transformers and filters. A user can create jobs and the system will show him/her a preview of the changes ordered before executing the job. The user can save the current job for future usages. They also provide a web application for monitoring and sharing analysis jobs and results.

- **Cloudingo** [U10] enables users to create a set of filters that will be used to remove duplicate entries from datasets. Users can pass the data through one or multiple filters that they defined and execute them manually or automatically. It also allows user to schedule a filter job that will curate data at regular intervals. When fetching data, cloudingo automatically runs base filter job to remove duplicates within existing data and new data.
- **Refier** [U11] automatically curates data by learning string similarity and deduce the optimal fuzzy matching rules for any domain or language. Refier needs a training set to learn about the rules, but if unavailable, then need to provide a small set of matching rules. Reifier uses "Apache Spark" for both distributed and stand-alone data.
- **IBM Infosphere Quality Stage** [U12] enables users to investigate, clean and manage their data. It uses both built in and user defined data quality rules to curate data automatically. It standardizes the data coming from different sources into a common format by removing duplicates and merging multiple systems into a single view.

B AUTOMATION IN FEATURE ENGINEERING (FE)

B.1 Research Contributions

Transformation-based Approach: One popular way of automating feature engineering is to apply different transformations on original features to synthesize new features. For example, Katz et al. [32] have identified some basic operators that can transform a feature or combine several to create a new feature. The intuition behind this transformation-based approach is the following: "highly informative feature often results from manipulations of elementary ones." Using these operators, Katz et al. [32] synthesized multiple candidate features that were fed into the training process, producing a candidate feature that is added to the model based on its empirical performance during the training. They have conducted extensive evaluation on 25 datasets and three classification algorithms and shown that they can achieve overall 20% classification-error reduction.

Being inspired by Katz et al. [32], van den Bosch [62] used three different categories of operators (unary, binary, and group-by-them) to transform old features into new ones. They also used a background classifier to predict the usefulness of a feature. Another contribution of van den Bosch [62] is to separate the feature generation and feature selection steps. As a result, feature selection can now be performed on both original and derived features, which was not possible in previous approaches.

Generation-based Approach: Kanter and Veeramachaneni [30] developed a Deep Feature Synthesis algorithm that can automatically generate features for relational datasets. Their algorithm follows relationships in the data to a base field, and then sequentially applies mathematical functions along that path to create the final feature. Given entities, their data tables, and relationships, they defined a number of mathematical functions that are applied to create features at two both the entity and relational levels. They optimized the whole pathway by implementing an autotuning process to help it generalize to different datasets.

Mountantonakis and Tzitzikas [46] proposed a generative method based on linked data for discovering, creating, and easily selecting valuable features to describe a set of entities in any ML problem. Linked Data refers to a method of publishing structured data while its ultimate objective is linking and integration. In this process, they first discover datasets and URIs that contain information for a set of entities. Then, they provide users a large number of potential features that can be created for these entities. Finally, they automatically produce a dataset for the features that were selected by the user. They evaluated this approach by performing a fivefold cross validation for estimating the performance of different models for the produced datasets.

Learning-based Approach: Some researchers have framed feature generation as a learning task and trained ML models to generate informative features. Khurana et al. [34] presented such an approach using **reinforcement learning (RL)** to automate FE). Their strategy is to train an agent on FE examples, enabling the agent to learn an effective strategy to explore available FE choices under a certain budget constraint. This exploration is performed on a directed acyclic graph called "transformation graph," representing relationships among different transformed versions of the data. Across a variety of datasets, they have shown that models produced by their system reduced median error rate by 25%.

Kaul et al. [33] presented a regression-based feature learning algorithm called "AutoLearn." They apply regression on each feature to predict the values of other feature and use this regression forecast to supplement additional information in each record. *AutoLearn* includes four major steps. First, prepossessing based on information gain is done to filter out less informative features. Second, using distance correlation, pairwise correlated features are defined and searched in the original feature space. Third, the original feature space is transformed into the new feature space by learning a predictive relation between correlated features. Finally, newly constructed feature space is constrained by selecting a subset of features based on stability and information gain. The authors have shown that, compared to original feature space, features learned through their model can improve prediction accuracy by 13.28% and 5.87% with respect to other top performing models.

B.2 Open Source and Commercial Systems

- **ExploreKit** [U20] is an automated feature generation kit that is essentially an implementation of the transformation-based approach introduced by Katz et al. [32].
- **FeatureTools** [U13] is an automated feature engineering framework based on the Deep Feature Synthesis method developed by Kanter and Veeramachaneni [30]. *FeatureTools* can effectively perform feature engineering for relational and temporal data. They also provide an R implementation named featuretoolsR [U18].
- **Featexp** [U14] is a python package for feature exploration that shows graphs and plots to help users better understand the feature space. Featexp bins a feature into equal population bins and shows the mean value of the dependent variable (target) in each bin, the trend of which can help users understand the relationship between the target and the feature. Users can compare feature trends to identify noisy features by looking at the number of trend direction changes and the correlation between train and test trend. These graphs/plots also help user to do feature debugging and leakage detection.
- **MLFeatureSelection** [U15] has broken feature selection into three parts. First, it selects a sequence from raw data that will potentially contain the best feature combination. Second, it creates feature groups from the raw data and iteratively update them and remove less important features. Eventually, this process yields a set of important features. Third, a coherent threshold is picked and all the features having higher coherence value are selected as best possible features.
- **Feature Engineering & Feature Selection** [U16] provides a guideline for feature engineering and feature selection techniques including why to use them, when to use them and when to use which algorithm.
- **FeatureHub** [U17] provided a collaborative framework for feature engineering and instantiate this idea into a platform. This platform administers collaboration among data scientists where users can register features in the database as well as reuse features written by other users. FeatureHub automatically scores the features so users can get real-time feedback for their features.

- **Feast** [U19] introduces a centralized feature store that can work as a foundation of features used by different projects. Feast allows users to use historical feature data to produce sets of features for training ML models, and makes feature data available to models in production by serving API and ensuring consistency between training and serving is preserved.

C AUTOMATION IN ALTERNATIVE MODELS EXPLORATION, TESTING AND VALIDATION

C.1 Automation in Hyperparameter Tuning

C.1.1 Research Contributions. There is a vast amount of literature regarding the automation of hyperparameter optimization within a machine learning model. Initially, grid search was tried to explore the space of hyperparameters but researchers immediately realized that the search space gets too large to find a good set of hyperparameters within a reasonable amount of time. Interestingly, it was found that random search often does a decent job for finding an optimal set of hyperparameters (Bergstra and Bengio [5]). Afterwards, researchers have tried **sequential model-based optimization (SMBO)** by utilizing computer clusters and GPU processors (Bergstra et al. [7]), and removing SMBO limitations (Hutter et al. [25]). Recent research shows that Bayesian optimization framework used for hyperparameter tuning provides promising results (Snoek et al. [58]). Some researchers have also tried reversed stochastic gradient descent with momentum (Maclaurin et al. [42]), which yields promising results, too. In this section, we will briefly review the research landscape of automating hyperparameter optimization.

Random Search-based Approach: Bergstra and Bengio [5] have shown that random search proves to be more efficient for hyperparameter optimization than grid search and manual search. Over the same domain, random search is able to find good or better models with a fraction of the computation time needed for pure grid search. They have also shown that for most datasets, only a handful of hyperparameters matter; but at the same time, the hyperparameters that actually matter are different for different datasets. Grid search suffers from poor coverage of dimensions that really matter, as it allocates too many trials to explore dimensions that matter less. In most cases, random search finds a better model while consuming less computational time.

Sequential Model-based Approach: Bergstra et al. [7] have shown that, with the current availability of powerful computer clusters and GPU processors, it is possible to run more trials. As a result, using an algorithmic approach to optimize hyperparameters can find better results than random search. They have introduced two greedy sequential methods to optimize hyperparameter for neural networks and **deep belief networks (DBN)**, and shown that random search obtains unreliable results for training DBNs. Their two approaches for sequential model-based optimization are **Gaussian Process (GP)** approach and Tree-structured Parzen Estimator approach.

Hutter et al. [25] identified three key limitations of SMBO that prevents it from being used for general algorithm configuration task. The limitations they defined are as follows: (1) SMBO only supports numerical parameter; (2) target algorithm performance is optimized by SMBO only for single instances; and (3) SMBO lacks mechanisms for terminating poorly performing target algorithm early. This work attempts to remove the first two of these SMBO limitations. Through the generalization of SMBO framework, they introduced four components and based on them, they defined two novel SMBO instantiations that are as follows: **Random Online Adaptive Racing (ROAR)** procedure and **Sequential Model-based Algorithm Configuration (SMAC)**. ROAR is a simple model-free instantiation of the general SMBO framework that uses their new instansification mechanism to iteratively compare its randomly selected parameter configurations against the current incumbent. SMAC can be viewed to be an extension of ROAR that selects configuration based on a model rather than doing it at random.

Another research that described a conceptual framework to optimize hyperparameters is the work by Bergstra et al. [6], where they propose a meta modeling approach to support automated hyperparameter tuning. They demonstrated quick recovery of state-of-the art results on several unrelated image classification tasks with no manual intervention. Their framework has four conceptual components: first, a null distribution specification language that describes the configuration distributions; second, a loss function that is the criteria this framework desires to minimize, and that maps a legal configuration sampled form the distribution to a real value; third, the hyperparameter optimization algorithm that takes as input the null distribution and historical values of the loss function and suggests configurations to try next; and fourth, a database that stores experimental history of already-tried configurations.

Bayesian Optimization-based Approach: Snoek et al. [58] have approached the problem of hyperparameter optimization through Bayesian optimization framework and identified good practices for Bayesian optimization of machine learning algorithms. Bayesian optimization is an iterative algorithm with two key ingredients: a probabilistic surrogate model and an acquisition function to decide which point to evaluate next. Snoek et al. [58] argued that a fully Bayesian treatment of the underlying GP kernel is preferred to the approach based on optimization of the GP hyperparameters proposed by previous work. For performing Bayesian optimization, they have selected Gaussian process prior as a prior over the function space and chosen an acquisition function to construct a utility function from model posterior. While describing the algorithm, they have taken into account the availability of multiple processor cores to run experiments in parallel, for which they proposed a sequential strategy. To compute Monte Carlo estimates of the acquisition function under different possible results from pending function evaluations, this strategy takes advantage of the tractable inference properties of GP.

In a similar manner, Swersky et al. [60] have proposed a multi-task Bayesian optimization approach to transfer knowledge within multiple Bayesian optimization frameworks. Applying well-studied multi-task Gaussian process models to Bayesian optimization framework is the basis for this idea. They adaptively learned the degree of correlation among domains by treating a new domain as a new task and then using this information to hone the search algorithm. To perform optimization of related tasks, they restrict their future observations to the tasks of interest; once they have enough observations to estimate the relationship between tasks, then other tasks will act as additional observations without requiring any further functional evaluation.

Gradient Descent-based Approach: Maclaurin et al. [42] have derived a computationally efficient process that will compute hyperparameter gradients using reversed stochastic gradient descent with momentum. They have shown that optimization of validation loss with respect to thousands of hyperparameters can be achieved through these gradients.

C.1.2 Open Source and Commercial Systems.

- **Hyperparameter Hunter** [U21] provides a wrapper for implementing machine learning algorithms by simplifying the experimentation and hyperparameter optimization process. It automatically uses results from past experiments to find optimal hyperparameters. It also eliminates "boilerplate" codes and can easily be integrated with other libraries.
- **Amazon Sagemaker** [U22] uses Bayesian Optimization to automatically discover optimal hyperparameters. It starts with a surrogate model and optimizes an special acquisition function called "Expected Improvement." In summary, it tries to define and explore a hyperparameter space using Bayesian Optimization and finds optimal hyperparameters that achieves the best model quality.
- **Advisor** [U37] is a black box hyperparameter optimization tool. It supports various black box optimization algorithms. Users can choose any algorithm based on their preference.

- **Hyperband** [U23] uses adaptive resource allocation and early stopping to speedup random search to automatically optimize hyperparameters. Predefined resources are allocated to randomly sampled configurations, and their hyperparameter optimization problem is formulated as a pure-exploration non-stochastic infinite-armed bandit problem.
- **BigML OptiML** [U24] uses Bayesian Optimization for hyperparameter tuning that is based on SMAC optimization technique. It sequentially tries a group of parameters depending on the results of training and evaluating models for those group of parameters. Users can also configure model search optimization metric and select from recommended top performing models.
- **Hyperboard** [U25] facilitates hyperparameter tuning by providing a visualization tool. Users can train models for different set of hyperparameters and plot the performances for each setting. Users can then select one set of hyperparameters by inspecting those plots and selecting the best performing configuration.
- **Google HyperTune** [U26] uses Bayesian optimization with Gaussian process as the optimization function and "Expected Improvement" as their acquisition function. In addition, they try to optimize all currently running jobs.
- **SHERPA** [U27] tried to optimize hyperparameters by using Bayesian Optimization via three options: GPyOpt, Asynchronous Successive Halving, and Population-Based Training. Additionally, they support parallel computation according to the user's need, and also provide a live dashboard for results.
- **Milano** [U29] provides automatic hyperparameter optimization and benchmarking while giving users the option to choose a cloud backend (Azkaban, AWS, or SLURM) of their choice. They also allow users to add search algorithms for bench-marking purposes.
- **Mind Foundry OPTaaS** [U30] provides automatic hyperparameter optimization using Bayesian Optimization. They optimize multiple metrics at a time in a secure way (does not access user's data or models) and can visualize probabilistic models.
- **BBopt** [U31] provides a blackbox optimizer for hyperparameter optimization. They support multiple algorithms for blackbox optimization such as random search, tree structured parzen estimator, Gaussian process, random forest, gradient boosted regression trees, serving, and extra trees.
- **Sigopt** [U32] uses an ensemble of Global and Bayesian Optimization algorithms so that they can choose the best optimization technique for the problem given by a user. They choose a suitable algorithm to explore the parameter space under the user's budget constraint.
- **Adatune** [U33] uses a gradient-based optimizer for hyperparameter tuning. Currently it only supports tuning of the hyperparameter "learning rate," but can be extended to parameters like "momentum" or "weight decay." They have used algorithms like "Hypergradient Descent," "Forward and Reverse Gradient-Based Hyperparameter Optimization," and "Scheduling the Learning Rate Via Online Hypergradients."
- **Gentun** [U34] uses genetic algorithms for hyperparameter tuning. They train a model by the set of hyperparameters defined by its genes. They have used a client-server approach to allow multiple clients to train their models, and cross-validation process is handled by a server. Offspring generation and mutation is also managed by the server.
- **Optuna** is an optimization software designed with "define-by-run" principle and is particularly the first of its kind. It divides the optimization process into two parts, sampling and pruning. For sampling, they use a blackbox optimization like Bayesian Optimization and grid search. For pruning, they use algorithms like hyperband where they cutoff trials that show little or no promise.

- **Test-tube** [U36] is a framework agnostic python library that parallelizes hyperparameter search across multiple CPUs and GPUs. They have used grid search and random search as optimization algorithm and user can choose any one of them as they like.

C.2 Automation in Alternative Models Exploration

C.2.1 Research Contributions. Alternative models exploration is another sub-area where the ML community has invested a lot of effort. These efforts are briefly summarized below.

Sequential Model-based Approach: One school of researchers adopted sequential model-based optimization for automatic selection of learning algorithm and tuning of corresponding hyperparameters. Thornton et al. [61] have dubbed this problem as *Combined Algorithm Selection and Hyperparameter Optimization* and approached it with Bayesian optimization, in particular, SMBO. SMBO first builds a model and uses it to determine candidate configuration of hyperparameters; then, it evaluates the loss and updates the model. They maximize "Positive Expected Improvement" as the acquisition function to decide the next useful configuration for evaluation. They combined the popular machine learning framework WEKA with Bayesian optimization to automatically select *good* instantiations of WEKA models, which they named as "Auto-WEKA."

In a similar spirit, Feurer et al. [18] extended the approach of Auto-WEKA [61] in the context of scikit-learn (another popular machine learning tool) and named it "auto-sklearn." They take into account the past performances on similar datasets that gives a warm-start to Bayesian Optimization and, from the models already considered by Bayesian optimization, they automatically construct ensembles through an efficient post-processing method. They focused their configuration space on base classifiers while excluding meta-models and ensembles that are themselves parameterized by one or more base classifiers, which resulted in fewer hyperparameters compared to Auto-WEKA.

Another research that used SMBO strategy to learn the optimal structure of CNN is the work by Liu et al. [38]. They start with a structured search space and search for convolutional cells instead of a full CNN. A cell consists of multiple blocks, where, a block is a combination operator (such as addition) applied to two inputs (tensors). Depending on training set size and desired runtime of final CNN, the cell structure is stacked. To search the space of cell structure, they proposed a heuristic search that gradually progresses from shallow models to complex models while pruning unpromising structures.

Gradient-based Approach: Zoph and Le [71] proposed a gradient-based method named "Neural Architecture Search" to find good neural-network architectures. They utilized a recurrent network to generate convolutional architectures, and used a controller to generate corresponding architectural hyperparameters. To maximize the expected accuracy of the sampled method, they have trained the neural network with a policy gradient method. They have also used set-selection type attention to enable controller to predict skip connections or branching layers.

Search Space-based Approach: Many researchers have framed automated model selection as a search problem. Zoph et al. [72] defined a search space (NASNet search space) with the property of transferability and searched for the best convolution layer for a small dataset. Then, they applied this cell to a larger dataset by stacking together multiple copies of the cell where each of them will have their own parameters. This way, they designed a convolutional architecture named "NASNet architecture." To improve generalization in the NASNet models, they have introduced a regularization technique called "Scheduled Drop Path."

Another research group that tried to improve the efficiency of NASNet [72] is Pham et al. [50], where, they proposed *Efficient Neural Architecture Search* for model discovery. They designed a controller to search for an optimal subgraph within a large computational graph. To select a subgraph that maximizes the expected reward in the validation set, they trained the controller with

policy gradient descent that minimized a canonical cross entropy loss. They forced child models to share parameters that allow them to achieve strong empirical performance.

Liu et al. [40] have designed a hierarchical search space where neural network architectures are represented through a hierarchical representation. At each level of the hierarchy, they allowed flexible network topologies. They kept a small set of primitives at the bottom level of the hierarchy and higher-level motifs are formed by lower level motifs. The final neural network is formed by stacking the motifs at the top of the hierarchy. Their work shows that, well designed search space can enhance the results of simple search methods. They have also presented a scalable variant of evolutionary search that improves the results.

Real et al. [52] used evolutionary algorithm to search the space of NASNet architectures [72]. They made two additions to the evolutionary search process; first, introducing age property to the tournament selection evolutionary algorithm to favour younger genotypes; second, implementing the set of mutations that NASNet search space would allow to operate. The mutation rules were restricted to only alter the architecture by randomly reconnecting and relabeling edges. They have shown that, specially in the earlier stages of evolution, their approach searched faster than reinforcement learning and random search.

Other Approaches: Lippmann et al. [37] introduced a DARPA program called D3M that aimed to automate model discovery system. They divided the program into two phases. In the first phase, the program built models for empirical science problems with prior complete data where each problem was be supplied with expert generated solutions. In the second phase, the program worked on unspecified and unsolved problems.

Baker et al. [2] tried to emulate human experts behavior of identifying and terminating suboptimal model configurations through the inspection of partial learning curve. They tried to parameterize learning curve trajectories by extracting simple features from the model architecture, training hyperparameters and early time-series measurements and used these features to train a set of regression models to predict the final validation performance of that configuration. They constructed an early stopping algorithm using these predictions and small model ensembles uncertainty estimates, which helped them to speedup both architecture search and hyperparameter optimization.

C.2.2 Open Source and Commercial Systems.

- **AutoKeras** [U47] is an open-source AutoML system for searching better neural architectures with guidance from Bayesian optimization technique. For optimization, they used a neural network as kernel and a tree structure as acquisition function.
- **AdaNet** [U49] is a tensorflow-based framework that require minimal expert intervention. AdaNet provides a framework for meta-learning better models while learning a neural network architecture. It learns the structure of a neural network as well as its weights.
- **Cloud AutoML** [U50] makes machine learning available to users even with limited machine learning knowledge. User can use AutoML to create a custom ML models that are tailored for the users data and integrate those models with the users applications.
- **PocketFlow** [U51] provides a framework that can compress and accelerate machine learning models. This framework has two components: learners and hyperparameter optimizers. Learners generate a compressed model from the original model with random hyperparameters, and the optimizer evaluates its accuracy and efficiency. After a few iterations, the model with best output is chosen as the final model.
- **Automl-gs** [U53] takes an input CSV file, infers the datatype of each attribute, and tries a ETL (extract, transform, and load) strategy for each attribute to create a statistical model. Model ETL and model construction functions are regarded as training scripts and once the

model is trained, training results are saved along with hyperparameters. This task is repeated until run count reaches a certain threshold and the best script is saved after each trial.

- **MLBox** [U55] is an automated machine learning library in python. MLBox can process, clean and format row data in distributed fashion. They use robust methods for feature selection and optimize the hyperparameters in high-dimensional space. For both classical and regression problems, they provide state-of-the art predictive models with interpretation.
- **Morph-net** [U57] learns optimal deep network structures during training. They added regularizers that target consumption of specific resources to activate sparsity. They used stochastic gradient descent to minimize summation of regularized loss.
- **Determined AI Platform** [U58] helps teams to collaborate by training models with greater speed, sharing GPU resources, and allowing users to build and train models at scale. Their hyperparameter optimizer uses distributed search of the parameter space that helps it quickly find optimal values for hyperparameters.
- **TransmogrifAI** [U60] is built on Scala and SparkML to automate machine learning pipelines. For alternate model selection, TransmogrifAI runs a tournament of different algorithms and automatically chooses the best one by using average validation error. To deal with imbalanced datasets, it appropriately samples data and recalibrates the prediction .
- **RemixAutoML** [U61] can guide users to a baseline starting model quickly. Once reached to baseline, alternative methods can be explored and compared with the baseline. It utilizes CatBoost, H2O, and XGBoost for machine learning automation jobs.

D AUTOMATION IN EVALUATION

D.1 Research Contributions

The machine learning community has spent significant efforts toward automation of evaluation and benchmarking for ML tasks. Shi et al. [55] conducted a benchmarking study of GPU-accelerated deep learning tools. Later, a benchmark study for Reinforcement Learning algorithm's continuous control tasks was reported by Duan et al. [15]. In a similar spirit, researchers have introduced publicly available benchmarking dataset suite for supervised classification method (Olson et al. [48]). In recent studies, benchmarking of hyperparameter optimization was conducted through surrogates (Eggensperger et al. [16]). In this section, we will briefly explore the research landscape of benchmarking and evaluation for machine learning tasks.

Shi et al. [55] aimed at benchmarking GPU-accelerated deep learning tools. They tried to document the running-time performance of some selected deep neural network tools on two CPU platforms and three GPU platforms. Afterwards, they also benchmark some distributed versions on multiple GPUs. From this benchmarking study, they concluded that no single tool consistently outperforms others that indicates further optimization opportunity.

Duan et al. [15] conducted a benchmark study of continuous control tasks for RL where they implemented and studied several RL algorithms in the context of general policy parameterizations. Their benchmark consists of 31 continuous control tasks that vary from simple to challenging tasks as well as contains partially observing and hierarchically structured tasks. Although several algorithm like TNPG, TRPO, and DDPG were efficient for training deep neural network policies, their poor performance on hierarchical tasks found during the study calls for further improvement.

Olson et al. [48] introduced a publicly available dataset suite named **Penn Machine Learning Benchmark (PMLB)** that initially had 165 publicly available datasets for supervised classification tasks. On the full set of datasets from PMLB, they conducted performance evaluation of 13 standard machine learning methods from scikit-learn. They have also assessed those benchmarking datasets

diversity based on their predictive performance and from the perspective of their meta features. To generalize ML evaluation, this work integrated real-world, simulated and toy datasets that make them a pioneer in the assembly of an effective and diverse set of benchmarking standards.

Eggensperger et al. [16] introduced the interesting idea of automatic benchmarking of hyperparameter optimization through surrogates that share the same hyperparameter space and feature similar response surface. They train regression models on the data of various performances of machine learning algorithm as the hyperparameter settings vary. Then, instead of running the real algorithm, they evaluate hyperparameter optimization methods using the models performance predictions. They have found that tree-based models can capture the performance of multiple machine learning algorithms reasonably well and can yield near real-world benchmarks without running the real algorithms.

D.2 Open Source and Commercial Systems

- **OpenML100** [U62] provides in-depth benchmarking of ML experiments on 100 public datasets and allows users to reproduce the results. They collect experiments created by users and evaluate them with different tools. During evaluation, every algorithm is evaluated with optimized hyperparameter-set and 200 random search iterations.
- **OpenML-CC18** [U63] benchmarks on 73 datasets. For each dataset, they run 1000 random configurations of random forest classifier where training time restricted to maximum three hours.

E SYSTEM AUTOMATION FOR ML

E.1 Research Contributions

Researchers have built various complex systems that can support data scientists performing different essential sub-tasks within an end-to-end machine learning pipeline. Some researchers used hybrid Bayesian and multi-armed bandit optimization systems (Swearingen et al. [59]), where other researchers have leaned toward black-box optimization (Golovin et al. [20]). Another line of work by Wang et al. [65] proposes a general programming model that provides a unified system architecture for both training and inference services. In this section, we will briefly review the research article that aim for ML system automation.

Swearingen et al. [59] introduced a machine learning service named ATM that can be hosted on any distributed computing infrastructure. ATM can support multiple users and search through multiple machine learning methods. They have proposed a hierarchical hyperparameter search space containing three levels. The combined hyperparameter search space for a given modeling method is defined by a **conditional parameter tree (CPT)**, and a subset of CPT is defined as a hyperpartition. A hyperpartition contains all choices on the path from root to leaf nodes and a set of tunable conditional hyperparameters is fully defined by the path. Swearingen et al. [59] have broken the problem of automatic machine learning process into two sub-problems. First, they select hyperpartition using the bandit learning strategy (specifically, a multi-armed bandit framework). Second, they tune the hyperparameters using Gaussian Process based on meta-modeling techniques within a hyperpartition.

Golovin et al. [20] introduced "Google Vizier," which is a service for black-box optimization. By default, they use *Batched Gaussian Process Bandits*. They used a Matern kernel with automatic relevance determination. "Expected improvement" was chosen as the acquisition function. With a random start, they used a proprietary gradient-free hill climbing algorithm to find local maxima of the acquisition function, and a Gaussian Process regressor was used for uphill walk. Vizier guides and accelerates the current study through prior studies' data by supporting a form of transfer learning.

Wang et al. [65] presented a system called "Rafiki" that provides distributed hyperparameter tuning for training service and online ensemble modeling for inference service. They proposed a general programming model to support popular hyperparameter tuning approaches like grid search, random search, and Bayesian optimization. To initialize new trials, they proposed a collaborative tuning scheme that leverages the model parameters of current top-performing training trials. To optimize overall accuracy and reduce latency, they have used reinforcement learning-based scheduling algorithm for the inference service.

E.2 Open Source and Commercial Systems

- **DotData** [U38] provides an automation system for supporting machine learning pipelines. It automatically prepares the given data for feature engineering, explore the feature space, and select relevant features. It derives an accurate predictive model by automatically exploring state-of-the-art algorithms and tuning their hyperparameters.
- **ATM** [U59] is a multi-data system for automated machine learning that was introduced by Swearingen et al. [59].
- **TPOT** [U39] is a tree-based optimization tool to automate machine learning pipelines developed by Olson et al. [47]. TPOT optimizes machine learning pipeline using GP by maximizing classification accuracy for a given supervised learning dataset. TPOT uses GP for evolving the pipeline operator sequences and parameters for maximizing the classification accuracy of the pipeline.
- **IBM AutoAI** [U40] provides an automated machine learning model building and evaluation system. AutoAI cleans raw data and categorizes them based on data type. AutoAI tests candidate algorithms against a small subset of the data to rank the algorithms. Gradually, they increase the size of the subset for most promising algorithms. At a high level, AutoAI explores the feature space while maximizing model accuracy using reinforcement learning.
- **Auto-Sklearn** Feurer et al. [18] have jointly automated learning algorithm selection and hyperparameter selection and introduced an automated machine learning toolkit called Auto-Sklearn [U41]. They mainly leveraged Bayesian Optimization, meta-learning and ensemble construction.
- **Azure Machine Learning** [U42] uses combination of Bayesian optimization and collaborative filtering to solve the meta learning task of machine learning automation. They take uncertainty into account by incorporating a probabilistic model that determines which model to try next.
- **Google Cloud AI platform** [U43] tests built-in algorithms for datasets submitted by the user. If any built-in algorithm performs well, then that algorithm is selected; otherwise, users can create a training application. This AI Platform Training also uses Bayesian optimization for hyperparameter tuning.
- **Amazon AWS service H2O.ai** [U44] is Amazon's solution to automate machine learning pipelines. Besides supporting end-to-end automation, it also offers visualization and machine learning interpretability functions.
- **Data Robot** [U45] finds a good starting point for hyperparameter optimization and users can iterate from there by further customizing learning rate, batch size, network architecture, and so on. Data Robot provides state-of-the-art benchmarks to support comparative analysis for the user.
- **Amazon Forecast** [U46] produces a forecasting model from time-series data capable of making future predictions. Their predictions are up to 50% more accurate than traversing time-series data alone.

REFERENCES

[1] Oliver Zeldin, Sameer Indarapu, Damien Lefortier, Zheng Chen, Sushma Bannur, Jurgen Van Gael, John Myles White, and Jason Gauci. 2018. *Introducing the Facebook Field Guide to Machine Learning Video Series*. Retrieved from https://research.fb.com/the-facebook-field-guide-to-machine-learning-video-series/.

[2] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. 2017. Accelerating neural architecture search using performance prediction. arXiv:1705.10823. Retrieved from https://arxiv.org/abs/1705.10823.

[3] Yoshua Bengio. 2012. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the Trade*. Springer, 437–478.

[4] Guy Berger. 2018. LinkedIn 2018 Emerging Jobs Report. Retrieved March 2, 2019 from https://economicgraph.linkedin.com/research/linkedin-2018-emerging-jobs-report.

[5] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* 13, (Feb. 2012), 281–305.

[6] James Bergstra, Daniel Yamins, and David Daniel Cox. 2013. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International Conference on Machine Learning (PMLR'15)*. 115–123.

[7] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*. 2546–2554.

[8] Dylan Cashman, Shah Rukh Humayoun, Florian Heimerl, Kendall Park, Subhajit Das, John Thompson, Bahador Saket, Abigail Mosca, John Stasko, Alex Endert, et al. 2018. Visual analytics for automated model discovery. arXiv:1809.10782. Retrieved from https://arxiv.org/abs/1809.10782.

[9] Girish Chandrashekar and Ferat Sahin. 2014. A survey on feature selection methods. *Comput. Electr. Eng.* 40, 1 (2014), 16–28.

[10] Hongju Cheng, Danyang Feng, Xiaobin Shi, and Chongcheng Chen. 2018. Data quality analysis and cleaning strategy for wireless sensor networks. *EURASIP J. Wireless Commun. Netw.* 2018, 1 (2018), 1–11.

[11] Xu Chu, Ihab F. Ilyas, Sanjay Krishnan, and Jiannan Wang. 2016. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 2201–2206.

[12] Xu Chu, John Morcos, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. 2015. KATARA: Reliable data cleaning with knowledge bases and crowdsourcing. *Proc. VLDB Endow.* 8, 12 (2015), 1952–1955.

[13] Janez Demšar and Blaž Zupan. 2012. Orange: Data mining fruitful and fun. *Informacijska Družba – IS* 6 (2012).

[14] Iddo Drori, Yamuna Krishnamurthy, Remi Rampin, Raoni de Paula Lourenco, Jorge Piazentin Ono, Kyunghyun Cho, Claudio Silva, and Juliana Freire. 2018. AlphaD3M: Machine learning pipeline synthesis. In *AutoML Workshop at ICML*.

[15] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. 2016. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the International Conference on Machine Learning*. 1329–1338.

[16] Katharina Eggensperger, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. 2015. Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*.

[17] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2018. Neural architecture search: A survey. arXiv:1808.05377. Retreived from https://arxiv.org/abs/1808.05377.

[18] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*. 2962–2970.

[19] Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. 2019. An open source AutoML benchmark. arXiv:1907.00909. Retrieved from https://arxiv.org/abs/1907.00909.

[20] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D. Sculley. 2017. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1487–1495.

[21] Laura Gustafson. 2018. *Bayesian Tuning and Bandits: An Extensible, Open Source Library for AutoML*. M.Eng. Thesis. Massachusetts Institute of Technology, Cambridge, MA.

[22] Daniel Haas, Sanjay Krishnan, Jiannan Wang, Michael J Franklin, and Eugene Wu. 2015. Wisteria: Nurturing scalable data cleaning infrastructure. *Proc. VLDB Endow.* 8, 12 (2015), 2004–2007.

[23] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2019. AutoML: A survey of the state-of-the-art. arXiv:1908.00709. Retrieved from https://arxiv.org/abs/1908.00709.

[24] Markus Hofmann and Ralf Klinkenberg. 2016. *RapidMiner: Data Mining Use Cases and Business Snalytics Applications*. CRC Press.

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

[26] Frank Hutter, Jörg Lücke, and Lars Schmidt-Thieme. 2015. Beyond manual tuning of hyperparameters. *Künstl. Intell.* 29, 4 (2015), 329–337.

[27] Ihab F. Ilyas, Xu Chu, et al. 2015. Trends in cleaning relational data: Consistency and deduplication. *Found. Trends Databases* 5, 4 (2015), 281–393.
[28] Zhi-wei Ji, Min Hu, and Jian-xin Yin. 2011. A survey of feature selection algorithm. *Electr. Des. Eng.* 19, 9 (2011), 46–51.
[29] James Max Kanter, Owen Gillespie, and Kalyan Veeramachaneni. 2016. Label, segment, featurize: a cross domain framework for prediction engineering. In *Proceedings of the 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA'16)*. IEEE, 430–439.
[30] James Max Kanter and Kalyan Veeramachaneni. 2015. Deep feature synthesis: Towards automating data science endeavors. In *Proceedings of the IEEE International Conference on Data Science and Advanced Analytics (DSAA'15)*. IEEE, 1–10.
[31] Shubhra Kanti Karmaker Santu, Parikshit Sondhi, and ChengXiang Zhai. 2017. On application of learning to rank for e-commerce search. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 475–484.
[32] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. 2016. Explorekit: Automatic feature generation and selection. In *Proceedings of the 2016 IEEE 16th International Conference on Data Mining (ICDM'16)*. IEEE, 979–984.
[33] Ambika Kaul, Saket Maheshwary, and Vikram Pudi. 2017. AutoLearn-automated feature generation and selection. In *Proceedings of the 2017 IEEE International Conference on Data Mining (ICDM'17)*. IEEE, 217–226.
[34] Udayan Khurana, Horst Samulowitz, and Deepak Turaga. 2017. feature engineering for predictive modeling using reinforcement learning. arXiv:1709.07150. Retrieved from https://arxiv.org/abs/1709.07150.
[35] Aaron Klein, Zhenwen Dai, Frank Hutter, Neil Lawrence, and Javier Gonzalez. 2019. Meta-surrogate benchmarking for hyperparameter optimization. arXiv:1905.12982. Retrieved from https://arxiv.org/abs/1905.12982.
[36] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J Franklin, and Ken Goldberg. 2016. ActiveClean: Interactive data cleaning for statistical modeling. *Proc. VLDB Endow.* 9, 12 (2016), 948–959.
[37] Richard Lippmann, William Campbell, and Joseph Campbell. 2016. An overview of the DARPA data driven discovery of models (D3M) program. In *Proceedings of the NIPS Workshop on Artificial Intelligence for Data Science*.
[38] Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2017. Progressive neural architecture search. arXiv:1712.00559. Retrieved from https://arxiv.org/abs/1712.00559.
[39] De-Rong Liu, Hong-Liang Li, and Ding Wang. 2015. Feature selection and feature learning for high-dimensional batch reinforcement learning: A survey. *Int. J. Autom. Comput.* 12, 3 (2015), 229–242.
[40] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. 2017. Hierarchical representations for efficient architecture search. arXiv:1711.00436. Retrieved from https://arxiv.org/abs/1711.00436.
[41] Bo Long, Jiang Bian, Olivier Chapelle, Ya Zhang, Yoshiyuki Inagaki, and Yi Chang. 2014. Active learning for ranking through expected loss optimization. *IEEE Trans. Knowl. Data Eng.* 27, 5 (2014), 1180–1191.
[42] Dougal Maclaurin, David Duvenaud, and Ryan Adams. 2015. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the International Conference on Machine Learning*. 2113–2122.
[43] Pedro Marcelino. 2018. Transfer learning from pre-trained models. *Towards Data Science* (2018).
[44] Zhuqi Miao, Shrieraam Sathyanarayanan, Elvena Fong, William Paiva, and Dursun Delen. 2018. An assessment and cleaning framework for electronic health records data. In *Proceedings of the 2018 Institute of Industrial and Systems Engineers Annual Conference and Expo (IISE'18)*.
[45] Amanda Minnich, Noor Abu-El-Rub, Maya Gokhale, Ronald Minnich, and Abdullah Mueen. 2016. ClearView: Data cleaning for online review mining. In *Proceedings of the 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. IEEE Press, 555–558.
[46] Michalis Mountantonakis and Yannis Tzitzikas. 2017. How linked data can aid machine learning-based tasks. In *Proceedings of the International Conference on Theory and Practice of Digital Libraries*. Springer, 155–168.
[47] Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. 2016. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 485–492.
[48] Randal S. Olson, William La Cava, Patryk Orzechowski, Ryan J. Urbanowicz, and Jason H. Moore. 2017. PMLB: A large benchmark suite for machine learning evaluation and comparison. *BioData Min.* 10, 1 (2017), 36.
[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.* 12 (2011), 2825–2830.
[50] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. 2018. Efficient neural architecture search via parameter sharing. arXiv:1802.03268.
[51] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2020. Snorkel: Rapid training data creation with weak supervision. *VLDB J.* 29, 2 (2020), 709–730.

[52] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. 2018. Regularized evolution for image classifier architecture search. arXiv:1802.01548. Retrieved from https://arxiv.org/abs/1802.01548.
[53] Benjamin Schreck and Kalyan Veeramachaneni. 2016. What would a data scientist ask? automatically formulating and solving predictive problems. In *Proceedings of the 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA'16)*. IEEE, 440–451.
[54] Razieh Sheikhpour, Mehdi Agha Sarram, Sajjad Gharaghani, and Mohammad Ali Zare Chahooki. 2017. A survey on semi-supervised feature selection methods. *Pattern Recogn.* 64 (2017), 141–158.
[55] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. 2016. Benchmarking state-of-the-art deep learning software tools. In *Proceedings of the 7th International Conference on Cloud Computing and Big Data (CCBD'16)*. IEEE, 99–104.
[56] Patrice Y. Simard, Saleema Amershi, David M. Chickering, Alicia Edelman Pelton, Soroush Ghorashi, Christopher Meek, Gonzalo Ramos, Jina Suh, Johan Verwey, Mo Wang, et al. 2017. Machine teaching: A new paradigm for building machine learning systems. arXiv:1707.06742. Retrieved from https://arxiv.org/abs/1707.06742.
[57] Micah Smith, Carles Sala, James Max Kanter, and Kalyan Veeramachaneni. 2019. The machine learning bazaar: Harnessing the ML ecosystem for effective system development. arXiv:1905.08942. Retrieved from https://arxiv.org/abs/1905.08942.
[58] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*. 2951–2959.
[59] Thomas Swearingen, Will Drevo, Bennett Cyphers, Alfredo Cuesta-Infante, Arun Ross, and Kalyan Veeramachaneni. 2017. ATM: A distributed, collaborative, scalable system for automated machine learning. In *Proceedings of the 2017 IEEE International Conference on Big Data (Big Data'17)*. IEEE, 151–162.
[60] Kevin Swersky, Jasper Snoek, and Ryan P. Adams. 2013. Multi-task bayesian optimization. In *Advances in Neural Information Processing Systems*. 2004–2012.
[61] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 847–855.
[62] Suzanne van den Bosch. 2017. *Automatic Feature Generation and Selection in Predictive Analytics Solutions*. Master's thesis, Faculty of Science, Radboud University.
[63] Joaquin Vanschoren. 2018. Meta-learning: A survey. arXiv:1810.03548. Retrieved from https://arxiv.org/abs/1810.03548.
[64] Juan Wang, Lin-lin Ci, and Kang-ze Yao. 2005. A survey of feature selection. *Comput. Eng. Sci.* 27, 12 (2005), 68–71.
[65] Wei Wang, Jinyang Gao, Meihui Zhang, Sheng Wang, Gang Chen, Teck Khim Ng, Beng Chin Ooi, Jie Shao, and Moaz Reyad. 2018. Rafiki: Machine learning as an analytics service system. *Proc. VLDB Endow.* 12, 2 (2018), 128–140.
[66] Yiren Wang, Dominic Seyler, Shubhra Kanti Karmaker Santu, and ChengXiang Zhai. 2017. A study of feature construction for text-based forecasting of time series variables. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 2347–2350.
[67] Ian H. Witten and Eibe Frank. 2002. Data mining: Practical machine learning tools and techniques with Java implementations. *ACM Sigmod Rec.* 31, 1 (2002), 76–77.
[68] Lei Xu, Shubhra Kanti Karmaker Santu, and Kalyan Veeramachaneni. 2019. MLFriend: Interactive prediction task recommendation for event-driven time-series data. arXiv:1906.12348. Retrieved from https://arxiv.org/abs/1906.12348.
[69] Aoqian Zhang, Shaoxu Song, Jianmin Wang, and Philip S Yu. 2017. Time series data cleaning: From anomaly detection to anomaly repairing. *Proc. VLDB Endow.* 10, 10 (2017), 1046–1057.
[70] Marc-André Zöller and Marco F. Huber. 2019. Survey on automated machine learning. arXiv:1904.12054. Retrieved from https://arxiv.org/abs/1904.12054.
[71] Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. arXiv:1611.01578. Retrieved from https://arxiv.org/abs/1611.01578.
[72] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2017. Learning transferable architectures for scalable image recognition. arXiv:1707.07012. Retrieved from https://arxiv.org/abs/1707.07012.

ONLINE RESOURCES

[U1] http://openrefine.org/.
[U2] https://www.trifacta.com/products/wrangler/.
[U3] https://github.com/Factual/drake.
[U4] https://clarity.cloud.tibco.com/landing/feature-summary.html.
[U5] https://sourceforge.net/projects/dataquality/).
[U6] https://winpure.com/.
[U7] https://www.trustradius.com/products/talend-data-quality/reviews.

[U8] https://dataladder.com/.
[U9] https://datacleaner.org/.
[U10] https://cloudingo.com/.
[U11] http://nubetech.co/technology/.
[U12] https://www.ibm.com/uk-en/marketplace/infosphere-qualitystage.
[U13] https://www.featuretools.com/.
[U14] https://github.com/abhayspawar/featexp.
[U15] https://github.com/duxuhao/Feature-Selection.
[U16] https://github.com/Yimeng-Zhang/feature-engineering-and-feature-selection.
[U17] https://github.com/HDI-Project/FeatureHub.
[U18] https://github.com/magnusfurugard/featuretoolsR.
[U19] https://cloud.google.com/blog/products/ai-machine-learning/introducing-feast-an-open-source-feature-store-for-machine-learning.
[U20] https://github.com/giladkatz/ExploreKit.
[U21] https://github.com/HunterMcGushion/hyperparameter_hunter.
[U22] https://aws.amazon.com/sagemaker/.
[U23] https://github.com/zygmuntz/hyperband.
[U24] https://bigml.com/api/optimls.
[U25] https://github.com/WarBean/hyperboard.
[U26] https://cloud.google.com/ml-engine/docs/using-hyperparameter-tuning.
[U27] https://github.com/sherpa-ai/sherpa.
[U28] https://indiesolver.com/.
[U29] https://github.com/NVIDIA/Milano.
[U30] https://www.mindfoundry.ai/mind-foundry-optimize.
[U31] https://github.com/evhub/bbopt.
[U32] https://sigopt.com/.
[U33] https://github.com/awslabs/adatune.
[U34] https://github.com/gmontamat/gentun.
[U35] https://github.com/optuna/optuna.
[U36] https://github.com/williamFalcon/test-tube.
[U37] https://github.com/tobegit3hub/advisor.
[U38] https://dotdata.com/.
[U39] https://github.com/EpistasisLab/tpot.
[U40] www.ibm.com/Watson-Studio/AutoAI.
[U41] https://github.com/automl/auto-sklearn.
[U42] https://azure.microsoft.com/en-us/services/machine-learning/.
[U43] https://cloud.google.com/ai-platform/.
[U44] https://aws.amazon.com/marketplace/solutions/machine-learning/data-science-tools.
[U45] https://www.datarobot.com/.
[U46] https://aws.amazon.com/forecast/.
[U47] https://github.com/keras-team/autokeras.
[U48] http://docs.h2o.ai/driverless-ai/latest-stable/docs/userguide/index.html.
[U49] https://github.com/tensorflow/adanet.
[U50] https://cloud.google.com/automl/.
[U51] https://github.com/Tencent/PocketFlow.
[U52] www.c3.ai.
[U53] https://github.com/minimaxir/automl-gs.
[U54] www.firefly.ai.
[U55] https://github.com/AxeldeRomblay/MLBox.
[U56] www.builton.dev/ml-apis.
[U57] https://github.com/google-research/morph-net.
[U58] www.determined.ai.

[U59] https://github.com/HDI-Project/ATM.
[U60] https://github.com/salesforce/TransmogrifAI.
[U61] https://github.com/AdrianAntico/RemixAutoML.
[U62] https://www.openml.org/s/14.
[U63] https://www.openml.org/s/98.
[U64] http://automl.chalearn.org/data.

Received October 2020; revised March 2021; accepted June 2021