A Level-wise Taxonomic Perspective on Automated Machine Learning to Date and Beyond: Challenges and Opportunities

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

Big data is ubiquitous across domains, and more and more stakeholders are choosing to use machine learning to get the most out of their data. This pressing need has inspired researchers to develop tools for Automated machine learning (AutoML), which is essentially automating the process of applying machine learning to real-world problems. The primary goals of AutoML tools are to provide methods and processes to make Machine Learning available for non-Machine Learning experts (domain experts), to improve efficiency of Machine Learning and to accelerate research on Machine Learning. But although automation and efficiency are some of AutoML's main selling points, the process still requires a surprising level of human involvement. A number of vital steps of the machine learning pipeline, including understanding the attributes of domain-specific data, defining prediction problems, creating a suitable training data set, and selecting a promising machine learning technique, still tend to be done manually by a data scientist or a machine learning engineer on an ad-hoc basis. Often, this process requires a lot of back-and-forth between the data scientist and domain experts, making the whole process more difficult and inefficient. Altogether, Automated Machine Learning systems are still far from a "real automatic system". In this review article, we present a level-wise taxonomic perspective on AutoML systems to-date and beyond, i.e., we introduce a new classification system with seven levels to distinguish AutoML systems based on their level of autonomy. With this goal in mind, we first start with a discussion on how an end-to-end Machine learning pipeline actually looks like and which sub-tasks of *Machine learning Pipeline* has indeed been automated so far. Next, we highlight the sub-tasks which are still done manually by a data-scientist in most cases and how that limits a domain expert's access to Machine learning. Then, we introduce the novel level-based taxonomy of AutoML systems and define each level according to their scope of automation support. Finally, we provide a road-map of future research endeavor in the area of AutoML required to further automate the end-to-end Machine Learning pipeline and discuss some important challenges in achieving this ambitious goal.

ACM Reference Format:
Shubhra Kanti Karmaker Santu, Md. Mahadi Hassan, Micah J. Smith, Lei Xu, ChengXiang Zhai, and Kalyan Veeramachaneni. 2018. A Level-wise Taxonomic Perspective on Automated Machine Learning to Date and Beyond: Challenges and Opportunities. *, , Article 111 (August 2018), 35 pages. https://doi.org/10.1145/1122445.1122456

Authors' addresses: Shubhra Kanti Karmaker Santu, SKS0086@auburn.edu, Auburn University + MIT LIDS, Cambridge, MA; Md. Mahadi Hassan, sibathasan@gmail.com, Auburn University, Auburn, AL; Micah J. Smith, micahs@mit.edu, MIT LIDS, Cambridge, MA; Lei Xu, leix@mit.edu, MIT LIDS, Cambridge, MA; ChengXiang Zhai, czhai@illinois.edu, University of Illinois Urbana Champaign, Urbana, Illinois; Kalyan Veeramachaneni, kalyanv@mit.edu, MIT LIDS, Cambridge, MA.

XXXX-XXXX/2018/8-ART111 $15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The twenty-first century has seen a massive increase in web-based systems, and this has itself resulted in the generation of "big data." Companies across all domains and applications are newly focused on exploiting this data in order to improve their products and services and understand their users' needs. 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 (5X growth) and ML engineers (12X growth) in order to better make sense of their data [4], and at the same time, both industry and academia have made *Artificial Intelligence* (AI) and *Machine Learning* (ML) a contemporary priority.

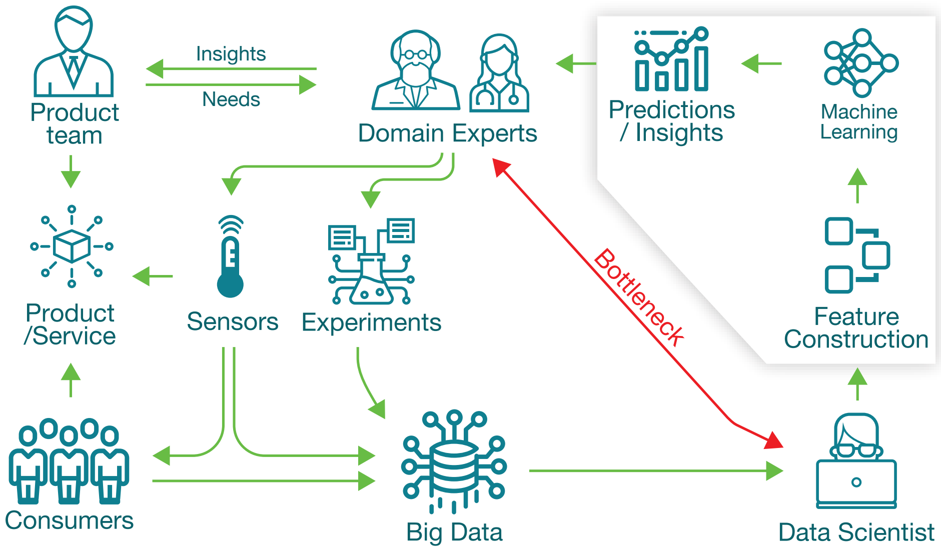
As more and more stakeholders are choosing to use machine learning to get the most out of their data, this pressing need has inspired researchers to develop tools for Automated machine learning (AutoML). AutoML is essentially a high-level abstraction paradigm for automating the process of applying machine learning to real-world problems. The primary goals of AutoML tools are to provide methods and processes to make Machine Learning available for non-Machine Learning experts (domain experts), to improve efficiency of Machine Learning and to accelerate research on Machine Learning. But although automation and efficiency are some of AutoML's main selling points, the process still requires a surprising level of human involvement. To better understand the problem, let us briefly discuss how an end-to-end machine learning process looks like today.

A typical machine learning process usually begins with a business or enterprise need. An organization's leadership team launches a product or service, and customers start interacting with it. Additionally, domain experts are appointed to dig deeper into product research and analyze customer behavior. The domain experts may deploy additional sensors and conduct new experiments to closely monitor different components of the system. All of these transactions, customer interactions, experiments and sensors produce a huge amount of data every day. It is impossible for the domain experts to manually mine knowledge and insights from all of this data — this is where the data scientists come in. Before moving forward, we would like to clearly define what we mean by the terms "data scientist" and "domain expert" and use this definition throughout the paper.

- **Domain Expert:** The person who knows the domain where ML is being applied to, but has minimal knowledge about how ML actually works.
- **Data Scientist:** The person who knows how ML actually works, but has minimal knowledge about the exact domain where ML is being applied to.

The data scientist first needs to clearly understand the data and the prediction goals from the domain experts. Then, the data scientist formulates computational prediction problems, extracts useful features from raw data, and selects a relevant ML model to solve the prediction problem. However, a number of vital steps of the machine learning pipeline, including understanding the attributes of domain-specific data, defining prediction problems, creating a suitable training data set, and selecting a promising machine learning technique, still tend to be done manually by a data scientist or a machine learning engineer on an ad-hoc basis. Often, this process requires a lot of back-and-forth between the data scientist and domain experts, making the whole process more difficult and inefficient. Once all these steps are done, existing AutoML models, which can automatically apply machine learning techniques to a carefully designed training / testing data-set with a well-defined prediction goal, can be readily deployed to train and test the learning outcome as well as draw valuable insights from the data. The data scientist then reports back to the domain experts and business leaders with these results and communicates the most interesting findings. Based on these insights, the domain experts and business leaders make informed decisions to improve the product and address any existing issues with the system. This whole process is visually presented in Figure 1.

Fig. 1. A typical case-scenario for end-to-end machine learning process. It involves multiple stakeholders bringing their unique expertise, tools, methods, data-sets and ability to modify the end product based on data driven insights and predictions.



As described above, the so-called "AutoML" systems we see today are still far from a real "automated" solution. In other words, an "Ideal AutoML" solution is expected to be an autonomous system which can perform *regular* data science tasks with minimal intervention from the humans, and current "AutoML" systems clearly have not yet attained that level of autonomy. In this review article, we present a new level-based taxonomy / classification system for AutoML solutions to-date and beyond, to summarize the different levels on automation support provided by them to complete a machine learning task. More specifically, we introduce a new classification system with seven different levels to distinguish AutoML systems based on their degree of automony. With this goal in mind, we first start with a discussion on how an end-to-end Machine learning pipeline actually looks like and which sub-tasks of *Machine learning Pipeline* has indeed been automated so far. Next, we highlight the sub-tasks which are still done manually by a data-scientist in most cases and how that limits a domain expert's access to Machine learning. Then, we introduce the novel level-based taxonomy of AutoML systems and define each level according to their scope of automation support along with some examples of existing AutoML systems at each autonomy level. Finally, we provide a road-map of future research endeavor in the area of AutoML required to further automate the end-to-end Machine Learning pipeline and discuss some important challenges in achieving this ambitious goal.

2 WHAT DOES A DATA SCIENTIST DO?

A quick inspection of the particular case-scenario of an end-to-end machine learning pipeline from Figure 1 reveals that *Data Scientists* often perform a well-defined set of functions in order to achieve the business analysis goals. Below, we formalize these steps performed. Of particular note is that the machine learning community and the systems community have spent a lot of efforts in the past decade into automating many of these steps, providing easy-to-use software interfaces and commercial systems to enable users to do some of these steps.

- **Task Formulation (TF):** As the first step of any *Data Science* endeavor, the *data scientist* interacts with the domain expert to understand the business problem they want to solve, examines what data is available, formulates a machine learning task which, if solved, could help in solving the broader business problem at hand. This process of formulating the task can involve multiple iterations of back and forth - each time considering multiple possibilities including checking if the required data is available, before honing in on a problem/task description. Currently, this is done manually, and is usually a long and arduous process. The process is ridden with mismatached expectations of what data could be useful vs what is available, communication gaps that arise due to lack of well defined common terminology, etc.

- **Data visualization, Cleaning and Curation (DCC):** This step is intertwined with all the other steps as we often get back to the original data we are working with to propose models, test our hypothesis and revise models accordingly. In general, a data scientist would identify relevant data, handle missing values, "*join*" multiple datasets and perform other functions for improving the quality of the data set. In doing this activity, a data scientist may also create intermediate visualizations to direct the process and aid in discussions. The *Data Mining*, *Systems* and *Database* community have spent significant effort to automate this step, which has been nicely summarized in Ilyas et al. [27] and Chu et al. [11]. In multiple approaches, machine learning itself is sometimes used to automatically do several aspects of curation, cleaning and linking of datasets. More recent approaches have focused on data cleaning challenges for specific domains including electronic health records (Miao et al. [43]), time series data (Zhang et al. [67]), online review mining (Minnich et al. [44]), and wireless sensor networks (Cheng et al. [10]). Another school of effort has primarily focused on interactive/active data cleaning (e.g., Chu et al. [12], Haas et al. [22], Krishnan et al. [35]). For a more detailed review of existing literature on Data visualization, Cleaning and Curation, refer to Appendix A.
- **Prediction Engineering (PE):** The task of prediction engineering primarily involves constructing and assigning labels to data points according to the goal prediction task and creating meaningful training and testing sets as a collection of <data_point, label> tuples. This task is also most of the time done manually by data scientists sometimes with the help of human data annotators. However, Kanter et.al. [29] tried to automate this process (a first work of this kind) by proposing a generalized 3 part framework - Label, Segment, Featurize (L-S-F), to address the growing demand for automated *Prediction Engineering*. The framework provides abstractions for data scientists to customize the *Prediction Engineering* process to unique prediction problems.
- **Feature Engineering (FE):** By the end of the first three steps, a data scientist would have defined a ML task (more commonly a predictive task), created the training/testing sets and started the process of building a model. As a first step in the process of building models, a *data scientist* would attempt to construct useful (informative) features from raw 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. [31], Kaul et al. [32], Khurana et al. [33], Mountantonakis and Tzitzikas [45], van den Bosch [61]). Multiple systems and open source libraries now exist with a focus on extracting useful features from data and generate feature matrices to construct the training set. For more details on the efforts for automated *Feature engineering*, refer to Appendix B.
- **Machine Learning (ML):** Once feature engineering is completed, the data scientist selects a suitable machine learning model based on the prediction task and starts training. These are basic machine learning techniques like decision tree, support vectors machines, linear regression, neural networks etc. which have current implementations across various software packages like scikit-learn [48], weka [65] etc. One crucial task associated with the training process is to optimize / tune any hyper-parameters associated with the learning model.
  - **Hyperparameter Tuning:** Machine learning models often contain multiple hyperparameters whose values are critical to obtaining good performance. Data scientists must tune these

hyperparameters to find models that work reasonably well. Related automation effort in this direction include Bengio [3], Bergstra and Bengio [5], Bergstra et al. [6, 7], Hutter et al. [25], Maclaurin et al. [41], Snoek et al. [57], Swersky et al. [59]. For a deeper discussion on *Hyperparameter Tuning* literature, refer to Appendix C.1.

- **Alternative Models Exploration, Testing and Validation (ATV):** A prediction task can generally be solved by multiple alternative ML techniques / pipelines. Currently, the works related to automation of selecting models, validating them and finalizing them are broadly categorized as a sub field called (so-called?) *AutoML* within the machine learning community. However, we decided to name this step as Alternative Models Exploration, Testing and Validation (ATV), as we view *AutoML* as an intelligent solution with a far broader scope of automating an end-to-end machine learning pipeline. Perhaps most widely studied in the past decade [1], major efforts in this direction include Feurer et al. [18], Swearingen et al. [58], Thornton et al. [60]. A number of commercial systems already exist for this task. Below we briefly describe the two sub tasks of ATV and point the reader to some automation systems, open source libraries and commercial systems.
  - **Alternative Models Exploration:** Exploring alternative models and finding their pros and cons is an important task for the data scientist. A recent DARPA-led initiative, known as "Data-Driven Discovery of Models" (D3M), focuses on alternative models exploration, i.e. the automation of "complete" data mining (Lippmann et al. [36]). It has inspired several works, 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 so-called "end-to-end" ML systems (Smith et al. [56]). In a similar vein, Olson et al. [46] developed a tree-based pipeline optimization tool, TPOT, that automatically designs and optimizes machine learning pipelines for a given problem domain, while Cashman et al. [8] studied how visual analytics can play a valuable role in the automated model discovery process. Corresponding with a recent rise in interest in deep neural networks, multiple researchers including Baker et al. [2], Liu et al. [37, 39], Pham et al. [49], Real et al. [51], Zoph and Le [69], Zoph et al. [70] 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 it on the testing set and report the statistical significance of these evaluations. Hence, the machine learning community has spent significant effort on benchmarking the evaluation results obtained by different machine learning models. Some recent efforts include benchmarking of deep learning software tools by Shi et al. [54], 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. [47]. More details about evaluation and benchmarking can be found in Appendix D.
- **Result Summarizing and Recommendation (RSR):** The final and perhaps, the most important job of a data scientist is to summarize all the findings and recommend the most useful / promising tasks to domain experts. For each task, recommendations can be made at the level of models, features, or computational overhead. This part in still done mostly manually without any systematic structure.

The different libraries / tools discussed so far, if deployed, typically make up one component within a larger system that aims to manage several practical aspects, such as parallel and distributed

[1]A quick search in google scholar with keywork AutoML will result in 1000s of paper and the top ones are cited more than 100 times

training, tuning, model storage, and even serving and deployment. Such large systems include ATM Swearingen et al. [58], Vizier Golovin et al. [20], and Rafiki Wang et al. [64], 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 tools developed so far to automate the machine learning process with details provided in the appendix for readers who are interested. It consists of routine sub-tasks that a data scientist does which have also been significantly studied for the purpose of automation. For each sub-task, Table 1 lists the notable research contributions, survey articles, open source systems and commercial systems available for that particular sub-task.

Table 1. A summary list of contributions in the broader field of Automated Machine Learning. For each sub-task, we listed the notable research contributions, survey articles, open source systems and commercial systems available for that particular sub-task. For more details on any work, refer to the Appendix.

Task	Research Contributions	Survey Articles	Open Source Systems	Commercial Systems
Data Visulaization, Cleaning and Curation	Miao et al. [43]	Ilyas et al. [27]	Open Refine [U1]	Trifacta Wrangler [U2]
	Zhang et al. [67] Minnich et al. [44]	Chu et al. [11]	Drake [U3] Open Source Data Quality and Profiling [U5]	TIBCO Clarity [U4] Winpure [U6]
	Cheng et al. [10]		Talented Data Quality [U7]	Data Ladder [U8]
	Krishnan et al. [35] Chu et al. [12] Haas et al. [22]			Data Cleaner [U9] Cloudingo [U10] Refier [U11] IBM Infosphere Quality Stage [U12]
Feature Engineering	Katz et al. [31]	Wang et al. [63]	FeatureTools [U13]	FeatureTools [U13]
	Kanter and Veeramachaneni [30]	Chandrashekar and Sahin [9]	featexp [U14]	
	Mountantonakis and Tzitzikas [45]	JI et al. [28]	MLFeatureSelection [U15]	
	van den Bosch [61]	Sheikhpour et al. [53]	Feature Engineering & Feature Selection [U16]	
	Khurana et al. [33] Kaul et al. [32]	Liu et al. [38]	FeatureHub [U17] featuretoolsR [U18] Feast [U19] ExploreKit [U20]	

[2]https://cloud.google.com/automl/

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

Table 1 – *Continued from previous page*

Task	Research Contributions	Survey Articles	Open Source Systems	Commercial Systems
Hyper-parameter Tuning	Bergstra and Bengio [5]	Vanschoren [62]	Hyperparameter Hunter [U21]	Amazon Sagemaker [U22]
	Snoek et al. [57]	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. [59]		adatune [U33]	
	Maclaurin et al. [41]		gentun [U34]	
			optuna [U35]	
			test-tube [U36]	
			Advisor [U37]	
Alternative Models Exploration	Thornton et al. [60]	He et al. [23]	AutoKeras [U47]	H2O Driverless AI [U48]
	Feurer et al. [18]	Elsken et al. [17]	AdaNet [U49]	Cloud AutoML [U50]
	Swearingen et al. [58]	Zöller and Huber [68]	PocketFlow [U51]	C3 AI Suite [U52]
	Lippmann et al. [36]		automl-gs [U53]	FireFly [U54]
	Zoph and Le [69]		MLBox [U55]	Builton [U56]
	Zoph et al. [70]		Morph-net [U57]	Determined AI Platform [U58]
	Liu et al. [39]		ATM [U59]	
	Liu et al. [37]		TransmogrifAI [U60]	
	Real et al. [51]		RemixAutoML [U61]	
	Pham et al. [49]			
	Baker et al. [2]			
Evaluation and Benchmarking	Shi et al. [54]	Gijsbers et al. [19]	OpenML100 [U62]	
	Gustafson [21]	Klein et al. [34]	OpenML-CC18 [U63]	
	Duan et al. [15]		AutoML Challenges [U64]	
	Olson et al. [47]			
	Eggensperger et al. [16]			
Systems	Swearingen et al. [58]		Scikit-learn	DotData [U38]
	Golovin et al. [20]		TPOT [U39] [46]	IBM AutoAI [U40]

Table 1 – *Continued from previous page*

Task	Research Contributions	Survey Articles	Open Source Systems	Commercial Systems
	Wang et al. [64]		Auto-Sklearn [U41]	Azure Machine Learning [U42]
			AlphaD3M [14]	Google Could AI platform [U43]
			ML Bazaar [56]	Amazoon AWS service [U44]
			PyTorch	Data Robot [U45]
				Amazon Forecast [U46]
				RapidMiner [24]
				Orange [13]

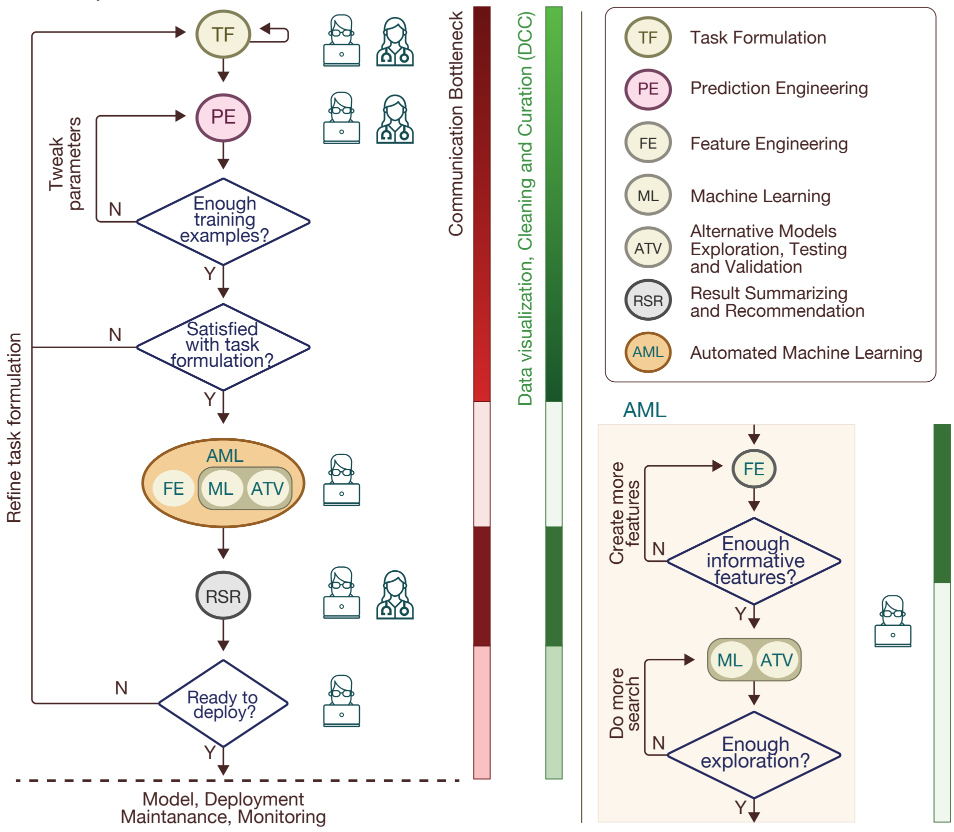
3 THE CURRENT FLOWCHART OF END-TO-END MACHINE LEARNING PIPELINE

The primary goal of *AutoML* endeavor is to reduce manual effort and thus, accelerate deployment of machine learning technologies in practice. Consequently, efforts have been made to develop systems that attempt to help reducing the manual work in some steps of the whole workflow of developing a machine learning application (see table 1 and the appendix for a detailed discussion). For example, DeepDive / Snorkel [50], is a general high-level workflow support system that can help users label and manage training data as well as provide high-level support for model selection. However, as mentioned in the previous section, developing ML solutions still involves a lot of manual work by humans, and it is important to address these limitations / bottlenecks in the current pipeline to design a “true” automated system. For better visualization of the current bottlenecks, we present in Figure 2 the current flowchart of end-to-end machine learning pipeline along with amount of manual work required by the data scientist, the role of domain experts during the process and the communication overhead between data scientist and the domain experts through the entire development cycle.

It is worth discussing Figure 2 in more details as it clearly highlights some key sub-tasks that need more attention and automation effort from the research community to further automate the entire machine learning pipeline currently in practice. As we can see, the biggest communication bottleneck currently happens during Task Formulation and Result summarizing and Recommendation sub-tasks, requiring significant amount of manual work. Prediction Engineering is another sub-task that sometimes requires a number of back-and-forth communication / clarification between the data scientist and the domain expert. These tasks are still largely unstructured and done manually by a series of trial / errors as well as meetings / feedback loops. On the other hand, the Feature Engineering (FE), Machine learning (ML) and Alternative Model Exploration, Testing and Validation (ATV) tasks are now fairly standard and the data scientist can use existing software / tools to execute them. It is important to note that domain experts currently have no / minimal access to FE, ML and ATV sub-tasks and as a result, they are kind of left out from the scene during this process. The role of the domain experts here is just to wait for the data scientist to perform their “magic tricks” and get back to them perhaps with some exciting results. Finally, once the business leaders agree to deploy a particular predictive model, the actual deployment, maintenance and monitoring is also done through a well-known standard process by the team of data scientists and software engineers.

In summary, detailed inspection of Figure 2 and the discussion above reveal that, the models and software developed so far have enabled or made significant progress towards the automation of the

Fig. 2. The Machine learning process cycle. The process cycle also highlights the points of interaction between the domain expert and the data scientists and the bottlenecks.

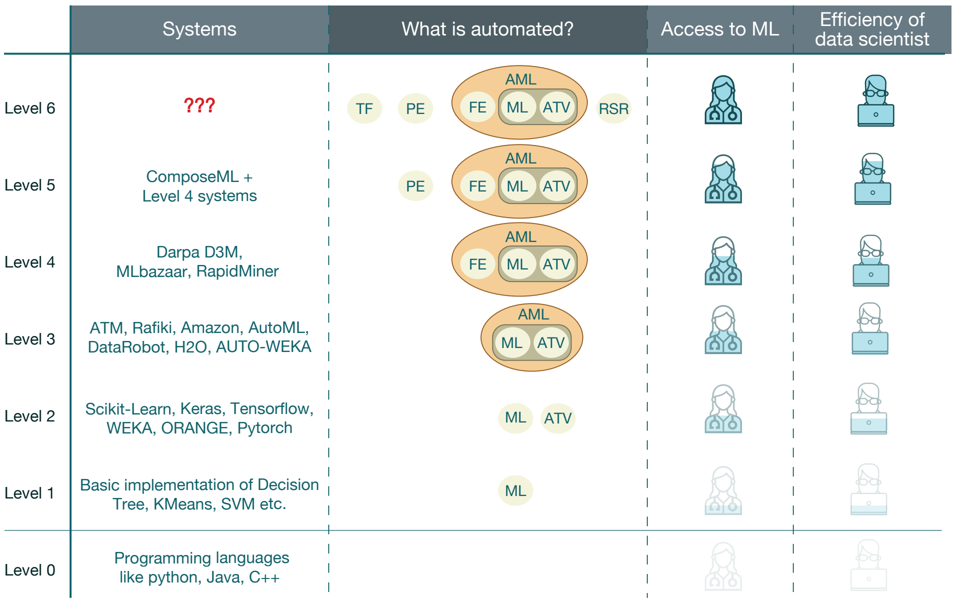

TF
PE
Tweak parameters
N
Enough training examples?
Y
Refine task formulation
N
Satisfied with task formulation?
Y
AML
FE
ML
ATV
RSR
N
Ready to deploy?
Y
Model, Deployment Maintanance, Monitoring
Communication Bottleneck
Data visualization, Cleaning and Curation (DCC)
TF Task Formulation
PE Prediction Engineering
FE Feature Engineering
ML Machine Learning
ATV Alternative Models Exploration, Testing and Validation
RSR Result Summarizing and Recommendation
AML Automated Machine Learning
AML
FE
Create more features
N
Enough informative features?
Y
ML
ATV
Do more search
N
Enough exploration?
Y

following essential tasks that need to be performed by a data scientist on a regular basis, namely, Data visualization, Cleaning and Curation (DCC), Machine Learning (ML), Feature Engineering (FE) and Alternative Models Exploration, Testing and Validation (ATV). However, the community has been reluctant to focus on the remaining three tasks, which are also very important, namely: *Task Formulation (TF)*, *Prediction Engineering (PE)*, and *Result Summarizing and Recommendation (RSR)*. One particular reason for such reluctance may be attributed to the human-centric nature of these three tasks. Indeed, these tasks demand significant human interaction in the process 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 recommendations to the users automatically. These missing pieces prevent the completion of a fully automated pipeline that can transform a data set into a useful predictive model and mine insights from the data.

4 THE NEW TAXONOMY OF AUTOML SOLUTIONS

To better characterize the quest for automation by the machine learning community, we first define a set of taxonomic levels which can represent the degree of intelligence / automation provided by the ML solutions developed so far. Here, we design the levels such that lower levels mean less automation / more manual work and higher levels mean more automation / less manual work. The levels also correlate with a domain expert's degree of direct access to a machine learning system without knowing the underlying intricate details of computer science and statistics. Figure 3

Fig. 3. Levels of automation for performing end to end machine learning endeavors. Level 0 requires one to develop and write software for the machine learning algorithms themselves. At this level one would write the basic algorithms like decision trees, topic modeling. At level 6 all tasks in executing a machine learning endeavor are automated including task formulation. The figure also identifies different systems at each level and where the current automation landscape stands. With each level of automation there is increased ability of domain expert to get involved as shown the color gradient in the domain expert. Increased automation has two simultaneous effects - first it enables domain experts to use machine learning and second it increases the productivity of the data scientist/machine learning expert. At the lowest level without the availability of the software that does basic, simple, yet core machine learning algorithms, domain experts cannot use machine learning at all. At the highest level domain experts can fully utilize machine learning. At the same time, manual work required by the data scientists significantly reduces as we go up in the levels.


Systems
What is automated?
Access to ML
Efficiency of data scientist
Level 6
???
TF
PE
AML
FE
ML
ATV
RSR
Level 5
ComposeML + Level 4 systems
PE
AML
FE
ML
ATV
Level 4
Darpa D3M, MLbazaar, RapidMiner
AML
FE
ML
ATV
Level 3
ATM, Rafiki, Amazon, AutoML, DataRobot, H2O, AUTO-WEKA
AML
ML
ATV
Level 2
Scikit-Learn, Keras, Tensorflow, WEKA, ORANGE, Pytorch
ML
ATV
Level 1
Basic implementation of Decision Tree, KMeans, SVM etc.
ML
Level 0
Programming languages like python, Java, C++

provides a visual illustration of the proposed taxonomic levels. We understand that there is no absolute meaning to these levels and the levels are merely a way characterize / organize machine learning solutions based on their degree of intelligence / autonomy.

- **Level 0 - No automation**
  All ML tasks are performed manually by hand-coded implementations and no automation is supported at this level. Machine learning researchers, who study and develop new machine learning models, mostly work at this level.
- **Level 1 - ML automated**
  Basic implementation of core Machine learning algorithms. This is the minimal level of automation supported by a system. Machine learning researchers mostly develop these tools and data-scientists are the primary users of tools at this level.
- **Level 2 - ML + ATV automated separately**
  These are automated tools heavily used by data scientists, however, still they have to do a lot of manual work. The domain experts still do not have any access to tools at this level.
- **Level 3 - ML + ATV automated jointly**
  Most of the work is done by the data-scientist, some of them through automated tools and rest of them by manual effort. The domain experts still do not have direct access to ML at this level.

- **Level 4 - ML + ATV + FE automated**
  This is the first level where the domain expert can start interacting (minimally) with the system, however, the amount of manual work by data-scientist remains significantly high.
- **Level 5 - ML + ATV + FE + PE automated**
  At this level, domain experts can comfortably interact with such systems with minimal help from the data scientist. Amount of manual work required at this level is also small.
- **Level 6 - ML + ATV + FE + PE + TF + RSR automated**
  Maximum level of automation supported, requires minimal manual work. Domain experts can easily use such systems by themselves.

To classify a machine learning tool into one of these seven levels, one just needs to inspect which of the Data Science sub-tasks (presented in section 2) has been automated as part of this tool. The Data Science sub-tasks themselves act as features here for classifying the AutoML tools and each feature will assume a binary value, where a 'yes' means automation is provided for that particular sub-task, whereas, 'no' means otherwise. For example, MLbazaar [56] provides automation for ML (Machine Learning), ATV (Alternative Models Exploration, Testing and Validation) and FE (Feature Engineering), but does not provide automation for TF (Task Formulation), PE (Prediction Engineering) and RSR (Result Summarizing and Recommendation). Thus, MLbazaar will be categorized as an AutoML system with level 4 automation. As another example, the popular data mining tool WEKA [65] provides automation for ML and ATV separately and thus, will be considered an AutoML system with level 2 automation, whereas, an extension of WEKA, i.e., AUTO-WEKA [60], provides automation for ML and ATV separately and thus, will be categorized as a level 3 tool.

Figure 3 clearly demonstrates the lack of automation tools at level 5 (only ComposeML) and level 6 (no solution yet), where the domain experts can contribute the most and data-scientists have to do minimal amount of work. The automation efforts so far have been primarily limited to Level 1 through 4, while domain experts can only effectively interact with ML systems at autonomy level 5 or 6. The major contribution of this paper is to highlight these gaps in current AutoML practice and draw the attention of the community towards this important yet relatively unexplored area.

5 TOWARDS LEVEL 6 AUTONOMY FOR AUTOML

How can we make the current machine learning process more efficient and automated? How can we relieve the communication bottleneck between data scientists and domain experts? One option is to enable the domain experts to directly engage with the fascinating world of big data without worrying about the underlying detailed knowledge of machine learning. Although this is a highly ambitious goal, if reached, it will expedite the implementation of machine learning systems allowing ML to appeal to a far broader audience. Imagine an intelligent agent that can guide domain experts in big data exploration and analytics tasks. In other words, think about an interactive intelligent agent, which can perform many of the routine tasks performed by a *Human* Data Scientist. Such a system would make big data analytics more appealing and accessible to business entities across various domains. At the same time, such system would increase the productivity of a human data scientist significantly by automating a big portion of their manual labor. We believe that only a level 6 AutoML tool / agent can provide such support, because on top of currently available automation support, it will also provide the support for automated task formulation (TF), prediction engineering (PE) and Result Summarizing and Recommendation (RSR).

The importance of *Task Formulation* (TF) and P*rediction Engineering* (PE) have also been substantiated by leading practitioners in the software industry. In popular tutorial materials, practitioners have discussed the value of TF and PE and their impact on actual use and business value of the resulting ML model [1]. For example, given the goal of predicting user enjoyment for a given mobile app — both hard to define and model — an alternative, simpler, goal like "number of sessions per week" can be defined, which is perhaps less relevant yet easier to define and model. According to these practitioners, the right prediction goal is often more important than the choice of algorithm, and a few hours spent at this stage in the process can save many weeks of work further downstream spent solving the wrong problem.

Previous work has laid some foundations in this area of *Prediction Engineering*. Schreck and Veeramachaneni [52] designed a formal language, called *Trane*, to describe prediction problems over relational data-sets, which allows data scientists to specify problems in that language. Another related work is MLFriend [66], where, the authors designed an interactive framework for defining useful prediction problems on event-driven time-series data and try to make it more appealing to domain experts. On an orthogonal yet related effort, Microsoft introduced an alternative training paradigm, called *Machine Teaching*, which facilitates training of machine learning models by non-experts [55]. The primary focus of *Machine Teaching* is to put a strong emphasis on the teacher (a non-expert in machine learning) and the teacher's interaction with data and training process. The authors have proposed a set of principles that lead to a better teaching process including a universal teaching language, "realizability" of all target concepts, rich and diverse sampling set, distribution robustness, etc. The authors pose that actively supervising the learning process by teaching can accelerate innovation and empower millions of new uses for machine learning models. These are all of course small steps towards the more ambitious goal of developing an agent at the intelligence level 6, yet they are limited by either their focus on a particular data science function (e.g. Machine Teaching primarily focuses on the training process, while MLFriend and Trane focus on defining the prediction task) or focus on a particular type of data (e.g. MLFriend is designed for event-driven time-series data) or lack of an user-centric prediction task recommender system (all of them are missing it).

How can we build an end-to-end automated pipeline of machine learning? In other words, how can we build an intelligent data science agent with an intelligence / autonomy level 6? In the following sections, we provide a road-map of future research endeavor in the area of AutoML required to further automate the end-to-end Machine Learning pipeline, i.e., to develop a fully functional level 6 intelligent agent. We will mainly discuss some important challenges and opportunities in achieving this ambitious goal.

6 CHALLENGES AND OPPORTUNITIES

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

Let's Do a Case Study: As most real world data comes in event-driven time series format and the primary goal of 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. To better demonstrate the challenges involved in realizing the level 6 AutoML agent, we picked a event-driven time series data set, **Flight Delay**,[4] which contains flight records for the United States from 2015, and generated prediction tasks from it. The data set has a timestamp associated with each event. We pick the time attribute, entity attributes, numerical attributes, and categorical attributes, while removing unsupported attributes like sets or natural language for simplicity. Table 2 shows the attributes we used to generate possible prediction tasks. Using this data set, we conducted an exploratory case study to understand how a level 6 AutoML agent can be materialized in a practical scenario.

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

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 categorical cancellation reason)
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)

6.1 Challenges in Task Formulation

Table 3. Operators

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

Our key observation is that across multiple domains, a prediction task is usually characterized by three components: an *outcome function*, a *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 where the most important translation from human intuition to problem specification happens, 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 by one definition could be quantified as *the number of times a customer visits the website during a week*. Here, outcome is *the number of visits to the website* and the `prediction window` is one week. Indeed, prediction task formulation is one of the most crucial steps of machine learning and practitioners from leading machine learning industry like Facebook have released a video series which specifically emphasizes this: "*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 the 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 general language in which the prediction tasks can be specified. Let us call this "Prediction Task Expression Language", or *PeTEL*. We argue that a complete *Task Expression Language* 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 like Trane [52] and MLFriend [66] have made progress on a reasonable design for Search Parameters and simple expressions for outcome function. However, expressing an outcome function still remains a significant challenge; specifically, how can we express an outcome function which is easily understandable by the domain experts with little or no experience in machine learning so that they can play with it? This motivation demands much more work to be done in this direction.

As with any language, PeTEL should provide constructs for variables, operators and functions. How should they look like? The 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 the implementations, 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. Take the following prediction goal for example:

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 Prediction Task Expression Language 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:

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

Although people with basic knowledge about structured query language may understand the meaning behind this expression, it is very hard for people with other backgrounds to understand the actual prediction task this PeTEL expression is trying to get across. However, if a level 6 AutoML agent translates the PeTEL expression into the following natural language form, it becomes clear to the user which prediction task is being proposed:

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 which needs to be addressed in order to make a level 6 AutoML agent appealing to a wider audience.

6.2 Challenges in Prediction Engineering for Identifying “Promising” Tasks

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

(1) **Task Validity**: While *PeTEL* can generate many different prediction tasks through combinations of operators and variables, not all combinations will yield a valid task. In fact, a large portion of them 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 and does not make any sense. Thus, one very important challenge for a level 6 AutoML agent is to figure out how to validate a proposed task and thus, filter the invalid ones.

(2) **User Preference**: Understanding which kinds of prediction tasks are generally liked by a particular user is very crucial to effectively identify *promising* prediction tasks. Capturing *User Preference* has two major benefits: 1) we can recommend highly relevant task to the users and 2) we can easily filter out tasks not liked by the user and save the corresponding AutoML execution time and computational resources. However, modeling *User Preference* in this context is tricky and often specific to the user and their domain. For example, an analyst at an airport authority may be more interested in predicting whether the airport they operate 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 depends heavily on modeling the user's preference accurately. In case of absence of a pre-trained user model, an active learning method to capture user's preferences seems to be very reasonable for this scenario. Specially, due to unavailability of sufficient training data in this case, the *Machine Teaching* perspective introduced by Microsoft can become really handy in this case [55].

(3) **Potential Business Value**: Another metric to filter out less *promising* tasks is to quantify their potential business value and discard the ones with low scores. Assuming that a particular prediction task can achieve reasonably high accuracy, would it really impact the business significantly? If not, it is not worth the effort to train a fancy machine learning model for the prediction task. *Potential Business Value* is often very hard to quantify and the best option is perhaps to ask domain experts who has long term experience to foresee the impact of such predictive models.

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

Finally, one can compute the overall promise of a prediction task by combining the four metrics above, i.e., User Preference, Task Validity, Business Potential Value and Availability of Sufficient Training Examples. Again, this is not an absolute standard by any means and is still an open problem.

6.3 Challenges in Result Summarizing and Recommendation (RSR)

As mentioned in section 2, *Result Summarizing and Recommendation* (RSR) is the most important job of a data scientist and this job in still done mostly manually without any systematic structure.

For realizing a level 6 AutoML agent, (although ambitious) this needs to be done automatically. The goal here is to rank prediction tasks based on their utility scores and present the top-k prediction tasks to the users (mainly the domain experts). Below we present a brief summary of the major challenges associated with the automation of *RSR* sub-task for a level 6 *AutoML* agent.

6.3.1 **Evaluation Challenges:** Prediction task recommendation essentially boils down to the core problem 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 the utility scores can be computed as a function of task specific metrics discussed in section 6.2 including pre-AutoML metrics like *User Preference*, *Task Validity* etc. as well as post-AutoML metrics like *Prediction Accuracy*, *Computation Time* etc. Some of these metrics can be computed automatically like, *Sufficient Training Examples*, *Prediction Accuracy*, while, some metrics like *User Preference*, *Potential Business Value* are highly dependent on the actual users, which in this case are domain experts or business leaders. Thus, utility appears to be complex metric composed of simpler metrics some of which are computed automatically vs some of which are collected through human annotations / feedback. However, it is unclear which type of feedback from the user — user rating, pairwise comparison, partial ranking, etc. — is the most appropriate way for gathering annotations.

6.3.2 **Representation Challenges:** The next challenge in designing a prediction task recommender system is how we represent the prediction tasks for machine learning setup. This means we need to transform the prediction task descriptions into a feature vector. Definitely the pre-AutoML and post-AutoML metrics can be used to compute meaningful numerical features. What about features related to task description? In other words, how do we convert the PeTEL expressions into a set of features suitable for machine learning? Should it be 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*? Given the promise of neural architectures, should we build a distributed representation / embedding for prediction task featurization? A detailed investigation of these questions are required to reach to a conclusion for representation.

6.3.3 **Learning challenges:** Given a particular *utility* metric and a particular feature representation for prediction tasks, a crucial challenge is to find out the best way to learn this *utility* metric. As with any learning system, it comes with basic challenges like 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?

As a starting point, we identify four different ways to perform the ranking of prediction tasks in order to provide users with recommendations. Below, we provide a brief summary of each of them.

(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* data set, we present some examples of highly recommended tasks in Table 4 that we curated through manual inspection. This static ranking list can be presented with or without the *utility* score of each task; however, it is currently unclear whether user experience will vary based on whether this *utility* score is shown or not.
    Static Ranking is desirable when ranking is performed based on well-defined numerical scores that can be automatically computed. For example, if we want the ranker to show tasks with high prediction accuracy only, then static ranking is sufficient to serve our purpose. Static Ranking may also work for cases where we have a very robust user model learned already from large historical data. For example, if we know the preferences of a particular user very well, Static Ranking may work reasonably well for that user.

Table 4. Example prediction tasks on *Flight Delay* data set. **PeTEL** is the structured representation of the problem, including the entity, attributes and operations. **Intermediate** is automatically generated natural language description of the problem. **Natural** is the human interpretation of the problem.

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 for 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 depart 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 most number of flights canceled because of weather tomorrow

(2) **Interactive Ranking with Feedback**: This setup is required for 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 about the user's preferences and revise the recommendation model to accommodate them (see [40] 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 very useful as a starting point for cases when the problem domain is not known historically. For example, a recommender system may be previously trained on car rental domain, but never been trained on truck rental domain. However, these two rental systems may share similarities as well as differences for the definition of utility of prediction task. One reasonable way to approach this problem is to transfer the knowledge learned from car rental domain and use it for initial recommendation for truck rental, an idea which is widely known as transfer learning and have recently gained much popularity [42]. Using transfer learning, one should can how users from truck rental domain react to such recommendations and evaluate its effectiveness.

(4) **Combining Interactive Ranking with Transfer Learning**: A more sophisticated way of recommending would be to start with a ranker learned from 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 car rental domain to provide recommendations in truck rental domain and at the same time, gather feedback from the users about the quality of the recommendations. Once the system gathers more information about truck rental domain, the recommender model can be revised to capture the available new information to better serve the target domain.

6.3.4 **Diversity challenges:** Another challenge for the recommender system is to ensure diversity in the list of recommended task. For example, consider the following four prediction tasks on *Flight Delay* data set:

- **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 $security\_delay$

Now, if we are to recommend two tasks from this set of four prediction tasks, 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 no clear understanding yet about how to achieve it.

7 HUMAN-AI INTERACTION IN LEVEL 6 AUTOML AGENT

In section 6, we presented the major challenges in materializing a level 6 AutoML agent by focusing on automation of the following three key functionalities: Task Formulation, Promising Task Identification for effective Prediction Engineering, and Result Summarizing and Recommendation. While automation is important for increased productivity and wider access, human analysts are integrally involved throughout the process. Ultimately, human analysts and decision makers must be convinced that the prediction tasks that the level 6 AutoML agent recommends are indeed useful and interesting in order to validate its effectiveness, suggesting several interesting challenges and opportunities in Human-AI interaction. Below we highlight some of them.

- **Problem Space Understanding:** Human analysts need to have some understanding of the problem space, i.e. the space of possible prediction tasks that can be generated by the level 6 AutoML agent, in order to gain context of the data science process. This closely relates with the problem of understanding the raw data that they have available and determining which are the useful tables and attributes, the frequency at which these data are collected and made available, and more. They should recognize that the level 6 AutoML agent can enumerate only a subset of this infinitely large problem space.
- **Prediction Task Understanding:** The level 6 AutoML agent may present a prediction task to human analysts at several stages to obtain their feedback. In what form should this task be presented? Multiple interfaces are possible, ranging from 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. It is likely that the understanding (and evaluation) of the prediction task will depend on the interface in which 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 evaluate 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:** Once one or more prediction tasks have been selected, end-to-end machine learning pipelines need to be developed and deployed to solve the tasks. This involves additional challenges, such as setting the free parameters of the task specification or adjusting prediction windows depending on how frequently data is collected and how quickly it is made available. These may involve additional rounds of communication between the human analyst and the level 6 AutoML agent.

8 THE LONG ROAD AHEAD

The AutoML community has provided us with powerful tools to execute any machine learning task and produce predictive models from them. However, defining the prediction problem itself and navigating the extremely large space of alternative prediction problems still remain as significant challenges for data scientists, let alone for domain experts and business leaders, who still struggle to access the power of big data analytics directly. Indeed, a significant portion of the machine learning process today require human involvement from a data scientist, making the whole process inefficient and inaccessible to a wider audience. In this review paper, we explained the gravity of this problem by introducing a new taxonomy for AutoML solutions with seven distinct autonomy levels and highlighting the current gaps and limitations in current machine learning practice. We then presented our vision for a more intelligent agent (level 6) to address this issue.

The primary benefits of a level 6 agent are twofold: 1) It will increase the productivity of data scientist by automating prediction task formulation and recommendation, 2) It will enable domain experts to directly access the fascinating world of predictive machine learning in an interactive fashion, thus potentially opening up new opportunities of applying ML that we might otherwise miss. In relation to this, we make the following key observations in this paper. First, formulating a prediction problem in the first place is a challenging task and currently there is no established standard on how to do it systematically. Second, the key technical challenge in realizing a level 6 AutoML agent essentially boils down to the design of an interactive prediction task recommender system, which engages the user actively into the loop.

We presented an exploratory case study, using a real-life data set to demonstrate the feasibility of actually realizing the hypothetical concept of a level 6 AutoML agent. Our initial exploration reveals that a large number of design and technical challenges must be addressed to reach the eventual goal of automated data science, specially, for automated task formulation, effective prediction engineering and recommendation of useful tasks. Another very important but orthogonal issue is the evaluation of the machine learning models, as fair evaluation is very critical to the success of AutoML. Indeed, fair evaluation of ML models depends on accurate design choices made throughout the entire *Data Science* process. For example, the training/ testing data-sets must be created in a way to ensure an accurate/robust model is learnt; benchmarking data-sets as well as evaluation test-beds should be built in order to assess the model accuracy/performance precisely and reliably; training labels should be created in a way which would ensure minimal annotation bias to the ML models; etc. In summary, it is very important for a level 6 AutoML tool to ensure an unbiased and representative model evaluation, as this is critical to the success of AutoML in general.

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

REFERENCES

[1] 2018. Introducing the Facebook Field Guide to Machine Learning video series. (May 2018). 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 preprint arXiv:1705.10823* (2017).

[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. (March 2018). 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. *Journal of Machine Learning Research* 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. (2013).

[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, and others. 2018. Visual Analytics for Automated Model Discovery. *arXiv preprint arXiv:1809.10782* (2018).

[9] Girish Chandrashekar and Ferat Sahin. 2014. A survey on feature selection methods. *Computers & Electrical Engineering* 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 Journal on Wireless Communications and Networking* 2018, 1 (2018), 61.

[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. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1952–1955.

[13] Janez Demšar and Blaž Zupan. 2012. Orange: Data mining fruitful and fun. *INFORMACIJSKA DRUŽBA- IS 2012* (2012), 6.

[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.

[15] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. 2016. Benchmarking deep reinforcement learning for continuous control. In *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 *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

[17] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2018. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377* (2018).

[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 preprint arXiv:1907.00909* (2019).

[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. *Proceedings of the VLDB Endowment* 8, 12 (2015), 2004–2007.

[23] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2019. AutoML: A Survey of the State-of-the-Art. *arXiv preprint arXiv:1908.00709* (2019).

[24] Markus Hofmann and Ralf Klinkenberg. 2016. *RapidMiner: Data mining use cases and business analytics applications*. CRC Press.

[25] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *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. *KI-Künstliche Intelligenz* 29, 4 (2015), 329–337.

[27] Ihab F Ilyas, Xu Chu, and others. 2015. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends in Databases* 5, 4 (2015), 281–393.

[28] Zhi-wei JI, Min Hu, and Jian-xin YIN. 2011. A survey of feature selection algorithm. *Electronic Design Engineering* 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 *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA).* IEEE, 430–439.

[30] James Max Kanter and Kalyan Veeramachaneni. 2015. Deep feature synthesis: Towards automating data science endeavors. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on.* IEEE, 1–10.

[31] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. 2016. Explorekit: Automatic feature generation and selection. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on.* IEEE, 979–984.

[32] Ambika Kaul, Saket Maheshwary, and Vikram Pudi. 2017. AutoLearn-Automated Feature Generation and Selection. In *Data Mining (ICDM), 2017 IEEE International Conference on.* IEEE, 217–226.

[33] Udayan Khurana, Horst Samulowitz, and Deepak Turaga. 2017. Feature Engineering for Predictive Modeling using Reinforcement Learning. *arXiv preprint arXiv:1709.07150* (2017).

[34] Aaron Klein, Zhenwen Dai, Frank Hutter, Neil Lawrence, and Javier Gonzalez. 2019. Meta-Surrogate Benchmarking for Hyperparameter Optimization. *arXiv preprint arXiv:1905.12982* (2019).

[35] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J Franklin, and Ken Goldberg. 2016. ActiveClean: interactive data cleaning for statistical modeling. *Proceedings of the VLDB Endowment* 9, 12 (2016), 948–959.

[36] Richard Lippmann, William Campbell, and Joseph Campbell. 2016. An Overview of the DARPA Data Driven Discovery of Models (D3M) Program. In *NIPS Workshop on Artificial Intelligence for Data Science.*

[37] Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2017b. Progressive neural architecture search. *arXiv preprint arXiv:1712.00559* (2017).

[38] De-Rong Liu, Hong-Liang Li, and Ding Wang. 2015. Feature selection and feature learning for high-dimensional batch reinforcement learning: A survey. *International Journal of Automation and Computing* 12, 3 (2015), 229–242.

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

[40] Bo Long, Jiang Bian, Olivier Chapelle, Ya Zhang, Yoshiyuki Inagaki, and Yi Chang. 2014. Active learning for ranking through expected loss optimization. *IEEE transactions on knowledge and data engineering* 27, 5 (2014), 1180–1191.

[41] Dougal Maclaurin, David Duvenaud, and Ryan Adams. 2015. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning.* 2113–2122.

[42] Pedro Marcelino. 2018. Transfer learning from pre-trained models. *Towards Data Science* (2018).

[43] Zhuqi Miao, Shrieraam Sathyanarayanan, Elvena Fong, William Paiva, and Dursun Delen. 2018. An assessment and cleaning framework for electronic health records data. In *2018 Institute of Industrial and Systems Engineers Annual Conference and Expo, IISE 2018.*

[44] 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.

[45] Michalis Mountantonakis and Yannis Tzitzikas. 2017. How linked data can aid machine learning-based tasks. In *International Conference on Theory and Practice of Digital Libraries.* Springer, 155–168.

[46] 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 2016.* ACM, 485–492.

[47] 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 mining* 10, 1 (2017), 36.

[48] 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. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[49] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. 2018. Efficient Neural Architecture Search via Parameter Sharing. *arXiv preprint arXiv:1802.03268* (2018).

[50] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2020. Snorkel: Rapid training data creation with weak supervision. *The VLDB Journal* 29, 2 (2020), 709–730.

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

[52] Benjamin Schreck and Kalyan Veeramachaneni. 2016. What would a data scientist ask? automatically formulating and solving predictive problems. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA).* IEEE, 440–451.

[53] Razieh Sheikhpour, Mehdi Agha Sarram, Sajjad Gharaghani, and Mohammad Ali Zare Chahooki. 2017. A survey on semi-supervised feature selection methods. *Pattern Recognition* 64 (2017), 141–158.

[54] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. 2016. Benchmarking state-of-the-art deep learning software tools. In *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*. IEEE, 99–104.

[55] Patrice Y Simard, Saleema Amershi, David M Chickering, Alicia Edelman Pelton, Soroush Ghorashi, Christopher Meek, Gonzalo Ramos, Jina Suh, Johan Verwey, Mo Wang, and others. 2017. Machine teaching: A new paradigm for building machine learning systems. *arXiv preprint arXiv:1707.06742* (2017).

[56] Micah Smith, Carles Sala, James Max Kanter, and Kalyan Veeramachaneni. 2019. The Machine Learning Bazaar: Harnessing the ML Ecosystem for Effective System Development. *arXiv e-prints* (2019), arxiv:1905.08942.

[57] 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.

[58] 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 *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 151–162.

[59] Kevin Swersky, Jasper Snoek, and Ryan P Adams. 2013. Multi-task bayesian optimization. In *Advances in neural information processing systems*. 2004–2012.

[60] 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.

[61] Suzanne van den Bosch. 2017. Automatic feature generation and selection in predictive analytics solutions. *Master's thesis, Faculty of Science, Radboud University* 3, 1 (2017), 3–1.

[62] Joaquin Vanschoren. 2018. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548* (2018).

[63] Juan Wang, Lin-lin Ci, and Kang-ze Yao. 2005. A survey of feature selection. *Computer engineering and science* 27, 12 (2005), 68–71.

[64] 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. *Proceedings of the VLDB Endowment* 12, 2 (2018), 128–140.

[65] Ian H Witten and Eibe Frank. 2002. Data mining: practical machine learning tools and techniques with Java implementations. *Acm Sigmod Record* 31, 1 (2002), 76–77.

[66] Lei Xu, Shubhra Kanti Karmaker Santu, and Kalyan Veeramachaneni. 2019. MLFriend: Interactive Prediction Task Recommendation for Event-Driven Time-Series Data. *arXiv preprint arXiv:1906.12348* (2019).

[67] Aoqian Zhang, Shaoxu Song, Jianmin Wang, and Philip S Yu. 2017. Time series data cleaning: From anomaly detection to anomaly repairing. *Proceedings of the VLDB Endowment* 10, 10 (2017), 1046–1057.

[68] Marc-André Zöller and Marco F Huber. 2019. Survey on Automated Machine Learning. *arXiv preprint arXiv:1904.12054* (2019).

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

[70] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2017. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012* 2, 6 (2017).

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

A AUTOMATION IN DATA VISUALIZATION, CLEANING AND CURATION (DCC)

A.1 Research Contributions

Framework Based Approach: One school of researchers primarily adopted a framework based approach for performing data visualization, cleaning and curation. For example, Miao et al. [43] introduced a data quality assessment and cleaning framework specifically for Electronic Health Records (EHR). They used the 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 have identified 23 critical data quality problem among which 16 were addressed by the proposed framework. In a similar effort, Zhang et al. [67] 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 repairs it; thus, it minimally change 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 $O(1)$. In order to enhance the accuracy of repairs in later iterations, the authors tried to learn the temporal nature of errors in anomaly detection. The paper 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. [44] introduced a cleaning process (called "ClearView") for online review mining purposes by categorizing noise into three categories. First is the syntactic noise which is usually misspelled words on which they perform two types of cleaning, character level cleaning, and word-level cleaning. 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 is the discrepancy between valid text sentiment and star rating. They have 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: the 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 the data quality. This strategy starts with computing the volume indicator of the data-set and determine if the cleaning process is necessary at all. Then, they execute the cleaning process via completeness, time-related, and correctness indicators.

Active Approach: Krishnan et al. [35] 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) towards a global solution which is a clean model. For the same amount of data cleaned, proposed optimizations in ActiveClean can improve model accuracy by up to 2.5x. 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 for them. 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, 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 about them.

- **Open Refine** [U1] allows users to clean and transform their data through a browser based interface. This is installed in clients machine and clients do not need to share their data unless contributing to the software. This system cleans data based on users decision.
- **Trifacta Wrangler**[U2] is a free cloud service that organize and structure the data automatically as soon as the data is imported into the system. The system then gives suggestion to users to do 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 which automatically resolve data dependency and calculates command list and their 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 which 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 Hadoop grid and create Hive tables.
- **Winpure** [U6] provides a data cleaning product which 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 which uncovers suspicious data and label them. Using those data, they build a classification model which predict match for new data source.
- **Data Ladder** [U8] provides a highly visual data cleaning application which automatically profiles data at the beginning. 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] has 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 data-sets. User 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 standalone 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. [31] have identified some basic operators which can transform a feature or combine several features 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. [31] synthesized multiple candidate features which were fed into the training process and a candidate feature is added to the model based on its empirical performance during the training. They have conducted extensive evaluation on 25 data-sets and three classification algorithms and shown that they can achieve overall 20% classification-error reduction.

Being inspired by Katz et al. [31], van den Bosch [61] 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 [61] 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 data-sets. 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 different levels: entity level and relational level. They optimized the whole pathway by implementing an autotuning process which helps it to generalize to different data-sets.

Mountantonakis and Tzitzikas [45] proposed a generative method, that is based on Linked Data, for discovering, creating and selecting, in an easy way, valuable features describing a set of entities for being used in any Machine Learning (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 data-sets 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 data-set for the features which were selected by the user. They evaluated this approach by performing a 5-fold cross validation for estimating the performance of different models for the produced data-sets.

Learning Based Approach: Some researchers have framed feature generation as a learning task and trained ML models to generate informative features. Khurana et al. [33] presented such an approach to automate feature engineering (FE) using reinforcement learning (RL). Their strategy is to train an agent on FE examples which enables 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" which represents relationships among different transformed versions of the data. Across a variety of data-sets, they have shown that models produced by their system reduced median error rate by 25%.

Kaul et al. [32] 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, pair-wise correlated features are defined and searched in the original feature space. Third, 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 which is essentially an implementation of the transformation based approach introduced by Katz et al. [31].
- **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 / 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, they select a sequence from raw data which will potentially contain the best feature combination. Second, from the raw data, they create feature groups 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 that users can get real-time feedback for their features.

- **Feast** [U19] introduces a centralized feature store which 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 model. They made feature data available to model in production via serving API and consistency between training and serving is preserved.

C AUTOMATION IN ALTERNATIVE MODELS EXPLORATION, TESTING AND VALIDATION (ATV)

C.1 Automation in Hyperparameter Tuning

C.1.1 **Research Contributions.** There exists 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. [57]). Some researchers have also tried reversed stochastic gradient descent with momentum (Maclaurin et al. [41]) 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 hyper parameter optimization than grid search and manual search. Over the same domain, random search is able to find good or better models within a fraction of computation time compared to pure grid search. They have also shown that for most data sets, only a handful of hyperparameters matter, but at the same time, the hyperparameters that actually matter are different for different data sets. 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 and as a result 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 (TPE) approach.

Hutter et al. [25] identified three key limitations of sequential model based optimization (SMBO) that prevents it to be used for general algorithm configuration task. The limitations they defined are: (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 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 which are: 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 which 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 which selects configuration based on a model rather than doing it at random.

Another research which described a conceptual framework to optimize hyperparameters is the work by Bergstra et al. [6], where they proposed a meta modeling approach to support automated hyperparameter tuning. They demonstrated quick recovery of state-of-the art results on several unrelated image classification task with no manual intervention. Their framework has four conceptual components: first, a null distribution specification language which describes the configuration distributions; second, a loss function which is the criteria this framework desires to minimize and it 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. [57] 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. [57] 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. [59] 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 used this information to hone the search algorithm. To perform optimization of related tasks, they restrict their future observations to the tasks of interest and 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. [41] 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 and 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 which is based on Sequential Model-based Algorithm Configuration (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 the currently running job.
- **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 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 (doesn’t access user’s data or models) and can visualize probabilistic models.
- **BBopt** [U31] provides a blackbox optimizer for hyperparameter optimization. They support multiple algoritms 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 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 gradient based optimizer for hyperparameter tuning. Currently it supports only tuning of the hyperparameter ‘learning rate’, but, can be extended to parameter 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 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 blackbox optimization like Bayesian Optimization and grid search. For pruning, they use algorithms like hyperband where they cutoff trials that shows 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 have invested a lot of efforts for automation, these 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. [60] have dubbed this problem as *Combined Algorithm Selection and Hyperparameter Optimization* (CASH) and approached it with Bayesian Optimization, in particular, Sequential Model-Based Optimization (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 [60] 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 data-sets which 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 classifier while excluding meta-models and ensembles that are themselves parameterized by one or more base classifiers, which resulted in fewer number of hyperparameters compared to Auto-WEKA.

Another research which used sequential model-based optimization (SMBO) strategy to learn the optimal structure of CNN is the work by Liu et al. [37]. 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 which gradually progresses from shallow models to complex models while pruning unpromising structures.

Gradient Based Approach: Zoph and Le [69] 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. [70] defined a search space (NASNet search space) with transferability property and searched for the best convolution layer for a small dataset. Then, they applied this cell to a larger data-set 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[70] is Pham et al. [49], where, they proposed *Efficient Neural Architecture Search* (ENAS) 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 which allow them to achieve strong empirical performance.

Liu et al. [39] 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 which improves the results.

Real et al. [51] used evolutionary algorithm to search the space of NASNet architectures [70]. 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 relabelling 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. [36] 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 sub-optimal 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 which require minimal expert intervention. AdaNet provides a framework for meta-learning better models while learning a neural network architecture. It learns both 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 intregate 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. Learner generates a compressed model from the original model with random hyperparameters and the optimizer evaluates its accuracy and efficiency. After few iterations, the model with best output is chosen as the final model.
- **Automl-gs** [U53] takes as input a CSV file, it then infers 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 which 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 which helps it to quickly find the optimal values for the hyperparameters.
- **TransmogrifAI** [U60] is built on Scala and SparkML that automates 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 data-sets, 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 job.

D AUTOMATION IN EVALUATION

D.1 Research Contributions

The machine learning community has spent significant efforts towards automation of evaluation and benchmarking for ML tasks. Shi et al. [54] 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. [47]). 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. [54] 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 which indicates further optimization opportunity.

Duan et al. [15] conducted a benchmark study of continuous control tasks for Reinforcement Learning (RL) where they implemented and studied several RL algorithms in the context of general policy parameterizations. Their benchmark consists of 31 continuous control tasks which varies 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. [47] introduced a publicly available data-set suite named Penn Machine Learning Benchmark (PMLB) which initially had 165 publicly available data-sets for supervised classification tasks. On the full set of data-sets from PMLB, they conducted performance evaluation of 13 standard machine learning methods from scikit-learn. They have also assessed those benchmarking data-sets 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 data-sets which 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 which 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 data-sets 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 data-sets. For each data-set, 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 to perform different essential sub-tasks within an end-to-end machine learning pipeline. Some researchers used hybrid Bayesian and multi-armed bandit optimization system (Swearingen et al. [58]), where other group of researchers have leaned towards black-box optimization (Golovin et al. [20]). Another line of work by Wang et al. [64] proposes a general programming model which provides a unified system architecture for both training and inference services. In this section, we will briefly review the research articles which aims for system automation for ML.

Swearingen et al. [58] introduced a machine learning service named Auto-Tuned Models (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 was defined by a conditional parameter tree (CPT), and a subset of CPT was 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. [58] have broken the problem of automatic machine learning process into two sub-problems. First, they select hyperpartition using bandit learning strategy specifically, multi-armed bandit (MAB) framework. Second, They tune the hyperparameters using Gaussian Process based meta-modeling technique 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 and "Eexpected 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. [64] 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 which leverages the model parameters of currently 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. They automatically prepare the given data for feature engineering, explore the feature space and select relevant features. They derive 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 which was introduced by Swearingen et al. [58].
- **TPOT** [U39] is a tree based optimization tool to automate machine learning pipelines which was developed by Olson et al. [46]. TPOT optimizes machine learning pipeline using genetic programming (GP) by maximizing classification accuracy for a given supervised learning data set. TPOT uses GP for evolving the pipeline operator sequences and parameters for classification accuracy maximization 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 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 feature space while maximizing model accuracy using reinforcement learning.
- **Auto-Sklearn** Feurer et al. [18] have jointly automated learning algorithm selection and hyper-parameter selection and introduced an automated machine learning toolkit called Auto-Sklearn [U41]. They tried to take advantage of 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 Could 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.
- **Amazoon AWS service H2O.ai** [U44] is the amazons's solution to automate Machine learning pipelines. Besides supporting end-to-end automation, it also offers visualization and machine learning interpretability (MLI) 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 more. Data Robot provides state-of-the-art benchmarks to enable comparative analysis for the user.
- **Amazon Forecast** [U46] produce a forecasting model from time series data which is capable of making predictions in the future. Their predictions are up to 50% more accurate than traversing time series data alone.