

AutoML: Towards an Integrated Framework for Autonomous Machine Learning

A Comprehensive and Synthesising Review of Concepts in AutoML Research and Beyond

DAVID JACOB KEDZIORA, Advanced Analytics Institute, University of Technology Sydney, Australia
KATARZYNA MUSIAL, Advanced Analytics Institute, University of Technology Sydney, Australia
BOGDAN GABRYS, Advanced Analytics Institute, University of Technology Sydney, Australia

Over the last decade, the long-running endeavour to automate high-level processes in machine learning (ML) has risen to mainstream prominence, stimulated by advances in optimisation techniques and their impact on selecting ML models/algorithms. Central to this drive is the appeal of engineering a computational system that both discovers and deploys high-performance solutions to arbitrary ML problems with minimal human interaction. Beyond this, an even loftier goal is the pursuit of autonomy, which describes the capability of the system to independently adjust an ML solution over a lifetime of changing contexts. However, these ambitions are unlikely to be achieved in a robust manner without the broader synthesis of various mechanisms and theoretical frameworks, which, at the present time, remain scattered across numerous research threads. Accordingly, this review seeks to motivate a more expansive perspective on what constitutes an automated/autonomous ML system, alongside consideration of how best to consolidate those elements. In doing so, we survey developments in the following research areas: hyperparameter optimisation, multi-component models, neural architecture search, automated feature engineering, meta-learning, multi-level ensembling, dynamic adaptation, multi-objective evaluation, resource constraints, flexible user involvement, and the principles of generalisation. We also develop a conceptual framework throughout the review, augmented by each topic, to illustrate one possible way of fusing high-level mechanisms into an autonomous ML system. Ultimately, we conclude that the notion of architectural integration deserves more discussion, without which the field of automated ML risks stifling both its technical advantages and general uptake.

Additional Key Words and Phrases: Automated machine learning (AutoML); Bayesian optimisation; Sequential model-based optimisation (SMBO); Combined algorithm selection and hyperparameter optimisation (CASH); Multi-component predictive systems; Predictive services composition; Neural architecture search (NAS); Automated feature engineering; Meta-learning; Concept drift; Dynamic environments; Multi-objective optimisation; Resource constraints; Autonomous learning systems; Artificial general intelligence (AGI)

1 INTRODUCTION

The field of data science is primarily concerned with the process of extracting information from data, often by way of fitting a mathematical model. Data science, as an umbrella term for techniques drawn from various disciplines, is agnostic as to who or what is driving that extraction. Indeed, while much effort has been dedicated to codifying effective workflows for data scientists [104], e.g. the Cross-Industry Standard Process for Data Mining (CRISP-DM) [61] and others [213], there is no inherent restriction that forces any phase of the process to be manually applied.

Certainly, in the modern era, one element of data mining and analysis is almost ubiquitously automated: model development, specifically model training. At one point in time, this was considered a novel advance, with computers only just becoming capable of running model-updating algorithms without human intervention. In fact, this form of automation was considered such a paradigm shift that it birthed the term ‘machine learning’ (ML) in the 1950s [304], provoked debate on its

Authors’ addresses: David Jacob Kedziora, Advanced Analytics Institute, University of Technology Sydney, Sydney, New South Wales, 2007, Australia, david.kedziora@uts.edu.au; Katarzyna Musial, Advanced Analytics Institute, University of Technology Sydney, Sydney, New South Wales, 2007, Australia, katarzyna.musial-gabrYS@uts.edu.au; Bogdan Gabrys, Advanced Analytics Institute, University of Technology Sydney, Sydney, New South Wales, 2007, Australia, bogdan.gabrYS@uts.edu.au.

“moral and technical consequences” [305, 360], and merged into the evolution of modern data analysis [342]. Advances since then, both fundamental and technological, have only cemented computational dominance of model development.

Now, more than 60 years beyond the dawn of ML, associated techniques and technologies have diffused through society at large. While advances in graphics processing units (GPUs) and big data architectures are credited with popularising deep neural networks (DNNs), abundant black-box implementations of the backpropagation method have also played their part. In effect, the need for human expertise to train complex inferential models has been lessened, and the last decade has consequently witnessed data science moving towards democratisation [42]. The 2012 journal article that is frequently credited with kicking off the DNN era sums it up well: “What many in the vision research community failed to appreciate was that methods that require careful hand-engineering by a programmer who understands the domain do not scale as well as methods that replace the programmer with a powerful general-purpose learning procedure.” [212]

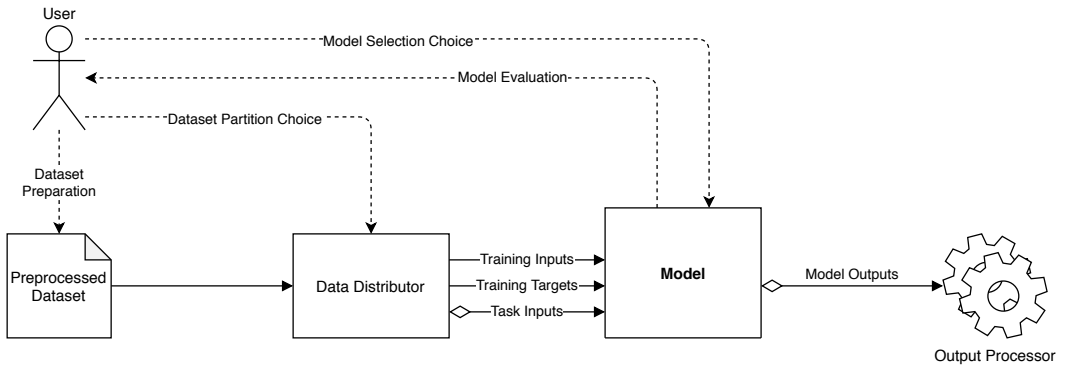


Fig. 1. Schematic of a predictive ML process, namely supervised learning, with heavy user involvement. Solid arrows depict dataflow channels. Dashed arrows depict control and feedback signals. Diamond arrow-tails depict conditional flows, distinguishing between model training and deployment phases.

Despite all this, common practice has not yet seen widespread mechanisation along the rest of the data science chain [76]. This can be seen by inspecting one typical workflow for running an ML task, as represented in Fig. 1. For the sake of simplicity, this task is one of supervised learning, but the depiction is similar for cases of delayed model-performance evaluation, e.g. unsupervised learning and learning based on reinforcement. In brief, given a specific choice of model, it is routine to train, validate and deploy the model on clean and informative datasets with ‘fit’ and ‘predict’ functionality that is provided by numerous coding packages. However, the typical data scientist still has to select a model, an evaluation metric and a training/validation/deployment strategy, all subject to human bias. As for data preparation, which combines cleaning and feature engineering, it is a notorious manual time-sink [67, 213].

The field of ‘automated machine learning’ (AutoML) [25, 154, 170, 339, 374, 387] has firmly established itself in recent years as a response to this; AutoML endeavours to continue mechanising the workflow of ML-based operations. It is motivated by the idea that reducing dependencies on human effort and expertise will, as a non-exhaustive list,

- make ML and its benefits more accessible to the general public,
- improve the efficiency and speed of finding ML solutions,
- improve the quality and consistency of ML solutions,
- enforce a systematic application of sound and robust ML methodologies,

- enable quick deployment and reuse of ML methodologies,
- compartmentalise complexity to reduce the potential for human error, and
- divert human resources to more productive roles.

In fairness, the field as a whole must also grapple with the risks of automation, including

- increased obscuration of ML technical debt [310],
- inappropriate or unethical usage as a result of ML illiteracy [42],
- interpretability issues in high-stakes contexts [293], and
- adverse socio-economic impacts such as job displacement [358].

In this article, we motivate and discuss synthesising major threads of existing AutoML research into a general integrated framework. Unlike other published reviews, we also broaden the scope to capture adjacent research that has been, to date, barely or not at all associated with the AutoML label, particularly in recent scientific literature. In effect, the aim of this review is to help inspire the evolution of AutoML towards ‘autonomous machine learning’ (AutonoML), where architectures are able to independently design, construct, deploy, and maintain ML models to solve specific problems, ideally limiting human involvement to task setup and the provision of expert knowledge.

Importantly, we do not profess the optimality of any architecture. Given the breadth and malleability of the field, it is unlikely that any one framework proposal will perfectly subsume all existing AutoML systems, let alone future advances. On the other hand, the merits and challenges of integration are best discussed with reference to concrete schematics. Thus, the literature review in this article is accompanied by the graduated development of a conceptual framework, exemplifying the potential interplay between various elements of AutonoML. As a necessity, Section 2 lays the initial groundwork for this example architecture by considering the fundamentals of ML, abstracting and encapsulating them as the lowest level of automatic operations.

The survey of AutoML begins in earnest within Section 3, which discusses the role of optimisation in automating the selection of an ML model/algorithm and associated hyperparameters. Section 4 then discards an assumption that the ML model need be monolithic, reviewing optimisation research for extended pipelines of data operators. This generalisation enables Section 5 to examine the optimisation of neural architectures specifically, while Section 6 focusses on the pre-processing elements of an ML pipeline, exploring the automation of feature engineering. Subsequently, Section 7 investigates how model search can be upgraded by meta-knowledge, leveraging information from external ML experiments, while Section 8 identifies the importance of ensembles and discusses how their management can be mechanised.

The paradigm of AutonoML is finally introduced in Section 9, defined by the ability to adapt models within dynamic contexts. This stimulates an examination within Section 10 of how the quality of any one solution should even be determined. Section 11 then examines the challenge of automating operations in low-resource settings, while Section 12 reviews efforts to reintegrate expert knowledge and user control back into autonomous systems. The survey ends with an acknowledgement in Section 13 of the drive towards one-size-fits-all AutonoML, drawing links to the quest for artificial general intelligence. Finally, Section 14 concludes with a discussion on the overarching technical challenges of integration, while remaining cognisant of the fact that understanding and fostering general engagement with resulting technologies are complex endeavours in their own right.

2 MACHINE LEARNING BASICS

At its core, standard ML is driven by the following premise: given a structured set of possible ‘queries’, \mathcal{Q} , and a corresponding space of possible ‘responses’, \mathcal{R} , there exists a mapping from one to the other that is maximally desirable for some intended purpose, i.e. an ML task. For instance,

one may seek a function from bitmaps to truth values for image classification, a function from historic arrays to predicted scalar values for time-series forecasting, or a function from sensor data to control signals for robotic operations. Naturally, for problems of interest, the optimal mapping is typically unknown across the entire space of \mathcal{Q} . Data scientists thus construct an ML model, $M : \mathcal{Q} \rightarrow \mathcal{R}$, to approximate this function, generalising where necessary, and then employ an ML algorithm to configure the model. If the algorithm is any good at its job, it will systematically acquire experience, learning from interactions with data so that the accuracy of the approximation improves. This process is called model development.

Importantly, an ML model may exist in one of many operational modes across the duration of its existence. Terminology and definitions vary greatly, but these include:

- Training – A mode in which the parameters of the ML model, typically in its initial state, are tuned by its associated ML algorithm. Any data that supports this process is called training data.
- Validation – A mode in which the accuracy of the ML model, given a selection of parameter values, is evaluated. The ML model converts incoming queries in \mathcal{Q} to corresponding responses in \mathcal{R} , which are compared against expected values. The result is typically used to tune the ‘hyperparameters’ of an ML model/algorithm and estimate predictive performance; see Sections 3 and 10, respectively. Validation data used in this process must be distinct from training data and must also be bundled with expected responses.
- Testing – A mode similar to validation, but where evaluated accuracy serves to judge the performance of a finalised ML model/algorithm for the ML task at hand. Testing data used in this process must be distinct from training/validation data and must also be bundled with expected responses.
- Deployment – A mode in which the ML model is put to practical use. It converts any incoming query in \mathcal{Q} to a corresponding response in \mathcal{R} ; these responses are used externally to fulfil the intended purpose of the ML model, e.g. decision-making.
- Adaptation – A mode in which the parameters of the ML model, previously trained and typically deployed, are re-tuned by its associated ML algorithm as a reaction to ML-model quality decreasing over time. This mode requires multiple tests and streamed data; see Section 9.

As a note, these modes are not necessarily independent. For instance, a rapid-deployment scenario may begin using an ML model for practical purposes while that model is still being trained. Regardless, any genuine AutoML system must facilitate all the above modes of operation.

To exemplify some of these modes, Figure 2 provides an example of how model training and ensuing model deployment can be managed by an automated system that acts in response to user requests. In this example, a data scientist is tackling the standard task of linear regression, having selected the simple linear function $y = ax + b$ and a least-squares estimator (LSE) as their ML model and ML algorithm, respectively. The training of the linear predictor proceeds chronologically as follows:

- (1) The linear model is initialised with default parameters, e.g. $(a, b) = (1, 0)$.
- (2–3) The user imports a training dataset into the ML system, which is collected in an inflow-data buffer, awaiting further orders. Each data instance contains a model input, x , and an expected output, Y . As befits a regression task, Y is a continuous variable.
- (4) The user orders the ML system via interface to begin training.
- (5–8) The interface directs the LSE to begin training. The LSE signals back its training requirements. The interface, processing this, signals the inflow buffer to transmit the training

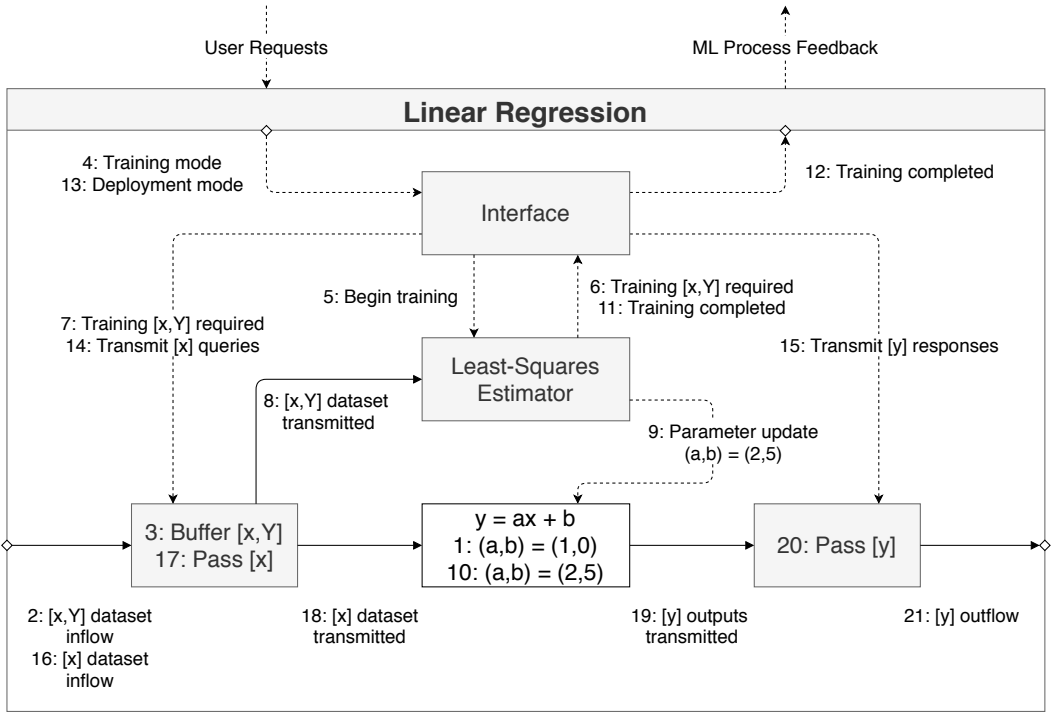


Fig. 2. An example of an ML system processing a linear regression task, i.e. training and deploying a linear model subject to a least-squares estimator (LSE). Steps are numbered chronologically and detailed in Section 2. Solid arrows depict dataflow channels. Dashed arrows depict control and feedback signals.

dataset to the LSE, which it does so. This is equivalent to calling a ‘fit’ method in a standard ML coding package.

(9–10) The LSE, according to its internal biases, calculates parameter values corresponding to the best linear fit for all provided (x, Y) coordinates. It signals the model to update its parameters to new values, e.g. $(a, b) = (2, 5)$.

(11–12) The LSE emits a training completion signal that is passed to the user.

The subsequent deployment and use of the linear predictor proceeds chronologically as follows:

(13) The user now orders the ML system via interface to deploy the trained model.

(14–15) The interface directs the inflow-data buffer to transmit incoming queries of the form x to the linear model. The resulting outputs of the form y are allowed to propagate onwards, e.g. to the user.

(16–18) The user imports a query dataset of x values into the ML system, which is directly passed to the linear model. This is equivalent to calling a ‘predict’ method in a standard ML coding package.

(19–21) The model calculates and emits associated response values, y . These are propagated onwards, e.g. to the user.

Figure 3 depicts a more complicated example of model development for an ML classification task, still managed by an automated system driven by user requests. Here, a data scientist has selected a multi-layer perceptron (MLP) as their ML model, training it via genetic algorithm (GA). Unlike the LSE of the previous example, the GA operates in an iterative manner and requires its parameter

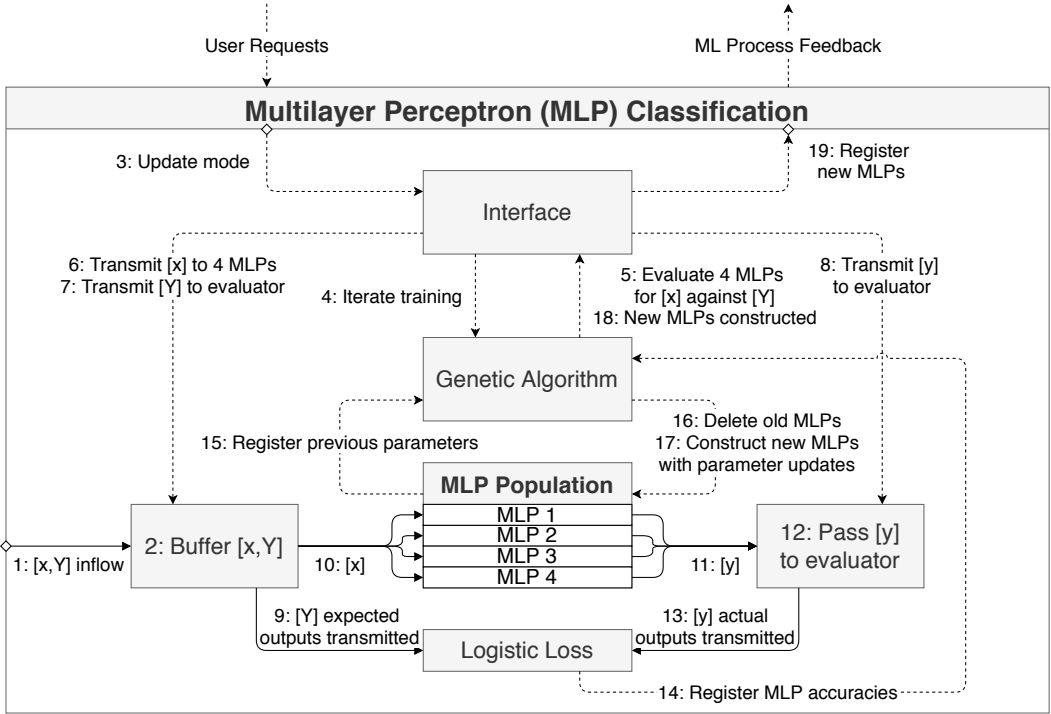


Fig. 3. An example of an ML system processing a multi-layer perceptron (MLP) classification task, i.e. updating a population of MLPs subject to a genetic algorithm (GA). A logistic-loss function is used to evaluate MLP accuracy. Steps are numbered chronologically and detailed in Section 2. Solid arrows depict dataflow channels. Dashed arrows depict control and feedback signals.

selections to be evaluated as part of the training process; the user has thus selected a classification-appropriate logistic-loss evaluator for this purpose. Due to the complexity of portraying a dynamic process within a static representation of ML architecture, Fig. 3 only shows one iteration of training. It assumes that the user has already acquired a set of four MLP instances, each with their own unique values for parameters, i.e. neural connection weights and biases. The update of the MLP ‘population’ proceeds chronologically as follows:

- (1–2) The user imports a training dataset into the ML system, which is collected in an inflow-data buffer, awaiting further orders. Each data instance contains a model input, x , and an expected output, Y . As befits a classification task, Y is a categorical variable.
- (3) The user orders the ML system via interface to begin updating the model.
- (4–5) The interface directs the GA to apply an iteration of training. The GA signals back its training requirements.
- (6–9) The interface directs the inflow buffer to transmit available training inputs, x , to the MLP population. The interface also signals expected and actual MLP outputs, Y and y , respectively, to be sent to the logistic-loss evaluator; Y is transmitted immediately.
- (10–13) For each MLP within the current population, the model transforms input x into categorical variable y , which is then compared against expected output Y . A logistic-loss score is determined per MLP.

- (14–17) The GA, registering the parameters and associated accuracy scores of each MLP model, selects parameters for a new population of MLPs. This selection is driven by elitism, crossover, mutation, and other principles relevant to the GA.
- (18–19) The algorithm emits a feedback signal to the interface, detailing the completed process. The interface propagates this signal back to the user or any other external ‘listeners’.

It should be noted that standard GA training continues to reiterate steps 4–19 for the same training inputs and expected outputs until some stopping criterion is met, such as a time limit. Subsequently, only the best performing model from the population is typically selected for deployment at the end of training, although an implementation may choose to keep the other models in memory for various reasons.

These are but two examples among a great diversity of ML tasks, yet they already reveal commonalities to be incorporated into any inclusive framework. Specifically, an ML model is essentially a data transformation, whether as complex as an MLP or as simple as a linear function. Correspondingly, an ML algorithm is a parameter controller for this data transformation; it has full knowledge of the model and, during a model development phase, can assign values to all of its parameters.

With these notions in mind, we now introduce the schematic shown in Fig. 4, which we propose to represent the foundation for an automated ML system: an ‘ML component’. Crucially, this concept should already be familiar to any AutoML developer, serving as an encapsulation that is amenable to substitution and, in advanced frameworks, concatenation; see Section 4. For simplicity, we will currently assume that a predictive system revolves around a solitary ML component. Additionally, while the schematic may appear complex, much of it is but a wrapper dedicated to providing a flexible interface for its two central elements: the data transformation and its associated parameter controller. Certainly, as the aforementioned LSE and GA examples show, not all depicted signals and dataflow channels are necessary, let alone active, in all contexts. Similarly, it is generality that motivates the use of a set-based container around the data transformation when coupling to a parameter controller. This is justified by Fig. 3 demonstrating that some ML algorithms work with multiple copies of an ML model. In essence, while we do not necessarily advocate Fig. 4 as the perfect way to structure an ML component, any alternative framework for truly generic AutoML should facilitate the same depicted processes, communications, and supported scenarios.

We begin detailing Fig. 4 by describing and justifying core interactions. Automated model development requires the parameter controller to be cognisant of all transformation properties, as well as possess the ability to tune parameters and, in some cases, be able to construct/delete transformation objects. Notably, these qualities are already implicit within many existing ML coding packages that treat a model object as an output of an algorithm object. However, the Fig. 4 schematic suggests that a data transformation and parameter controller should inherently be on equal footing; there is no theoretical reason why an ML model cannot be initialised alongside an ML algorithm using default values, e.g. the identity function $y = 1x + 0$ for linear regression. Even grown/pruned models with a variable number of parameters, sometimes called constructivist/selectivist models [119], have initial states. Default values may not be predictively powerful, but at least this representation supports the ‘anytime’ validity of an ML model that is required by rapid-deployment AutonoML; see Section 9.

Now, standard ML assumes that a data scientist is responsible for plugging in a combination of ML model and ML algorithm that is well suited to their objective and data, before driving the ML component through the various high-level operating modes described earlier. Each phase of operation may involve a different way of directing data around the component. For instance, certain ML algorithms may need to operate on data during initial training, such as the LSE portrayed in

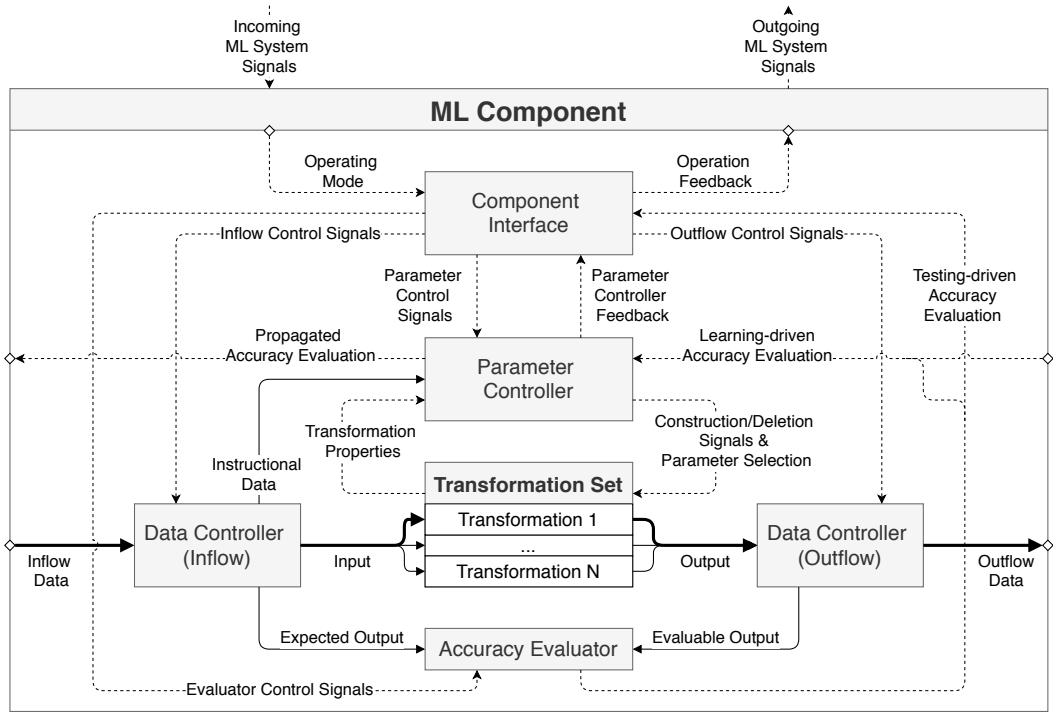


Fig. 4. A general schematic of an ML component, wrapped around a data transformation and its parameter controller. A component interface communicates with external systems and the parameter controller, relaying directions to a pair of data controllers. The data controllers channel inflow and outflow data according to model development/deployment needs, while an accuracy evaluator assesses transformation quality based on supervised-learning data. Accuracy evaluations are directed to the parameter controller or the interface depending on whether they are sought as part of a learning or testing process. Solid arrows depict dataflow channels; thick arrows denote dataflow during deployment. Dashed arrows depict control and feedback signals.

Fig. 2, while ML-model outputs may need to be held back from a user prior to a deployment phase, as exemplified by the GA-based training shown in Fig. 3. The schematic in Fig. 4 therefore includes an inflow-data controller (IDC) and an outflow-data controller (ODC) to disallow or distribute dataflow as required. In turn, these requirements are dictated by a component interface, which interprets instructions from external systems, or the user, and then translates them into lower-level directives that are disseminated as control signals. The interface also returns any relevant operational feedback to the source of external requests.

A generic ML component must additionally support an accuracy evaluator, allowing the quality of a data transformation to be assessed by comparing its outputs against expectation whenever available. In the Fig. 4 schematic, this module is wrapped up within the ML component and managed by the interface. As with the parameter controller and associated transformation, the contents of this evaluator vary by ML task, e.g. the logistic-loss calculator in Fig. 3. Moreover, it is possible that an ML component need not always instantiate the accuracy evaluator; certain ML models/algorithms seek novel patterns in inflow data for which the traditional concept of accuracy makes no sense.

Returning to the topic of operating modes, what a user wants at a high level may not always project neatly into the lower level of an ML component. Even variations between ML models/algorithms can result in divergent processes, as demonstrated by the differences in training between the aforementioned LSE and GA examples; the latter employs calls to an accuracy evaluator as part of its internal learning procedure. This is why the component interface needs to translate concepts such as training and deployment, with regard to specific classes of ML model/algorithm, into combinations of atomic low-level directives. The following is one such proposal:

- ‘Update’ – Request the parameter controller to continually tune the transformation set until some stopping criteria is reached. Direct the IDC to distribute inflow data in any way that supports the parameter-controller algorithm.
- ‘Validate’ – Direct the IDC to pass input data to the transformation set, then direct the IDC and ODC to distribute expected output and actual output, respectively, to the accuracy evaluator.
- ‘Propagate’ – Direct the IDC to pass input data to the transformation set, then direct the ODC to propagate the output further downstream.

In any implementation, these calls will likely be paired with further arguments to, for example, specify which batches of inflow data should be used for each process.

Given this basis set of low-level operations, a high-level request for model deployment is the most trivial to translate, mapping simply to a ‘propagate’ call. Steps 13–15 for the LSE example in Fig. 2 demonstrate the relevant signals, while steps 16–21 depict deployed-model dataflow. The only implementational challenge is ensuring that the IDC can filter inflow data appropriately and, in the case of transformation plurality, identify the right ML model to feed with input data. All of this can be handled by the interface relaying signals from the parameter controller over to the IDC, specifically detailing the properties of the transformation set.

Similarly, high-level desires to validate or test an ML model, usually differing by which subsets of available data are used in the process, map directly to the ‘validate’ directive. The accuracy evaluator is responsible for this procedure, comparing transformed inputs with expected outputs. Both higher-level modes, i.e. validation and testing, usually serve to inform the user about the quality of an ML model/algorithm, which is why Fig. 4 depicts ‘testing-driven’ evaluations being sent to the component interface for external dissemination. This contrasts with the case of ‘validate’ calls being part of model development, wherein accuracy evaluations are ‘learning-driven’ and sent directly to the parameter controller so as to advise the ML algorithm in tuning ML-model parameters. This is an important nuance; rolling validatory mechanisms into the learning process can result in dramatically different training runs, even with the same ML model/algorithm/evaluator [117]. Different ways of mixing the same underlying ML algorithm with learning-driven accuracy evaluation can potentially be encapsulated by an extra hyperparameter; see Section 3.

As an additional point of note, the high-level validation of an ML model/algorithm is typically involved, commonly requiring repeat training/validation runs on multiple resamplings of inflow data. Two examples of this process are k-fold cross-validation and out-of-bootstrap validation. Alternatively, multiple stochastic iterations can be avoided with methods that construct a representative sample [50–52], i.e. a subset that preserves the statistics of the superset, but these require additional dataset analysis. Again, we do not advocate for any particular implementation here; developers may decide whether responsibility for the management of a cross-validatory process, for example, falls to the component interface or to a higher level. However, the cleanest delegation of responsibility suggests that the interface should only direct operations per received data sample, with the management and aggregation of all resamplings taking place outside of the ML component, as elaborated in high-level schematics within Section 3.

Ultimately, operation modes that involve model development, i.e. the ‘update’ directive, are where complexities truly arise. For a general ML system, the interface must be flexible enough not just to direct the parameter controller to tune the transformations but also to understand what it needs for that tuning procedure. This communication via parameter control signals and parameter controller feedback is demonstrated by steps 5–6 for the LSE example in Fig. 2. In general, training requirements can vary dramatically, given that numerous parameter selection strategies exist, with differences depending on the properties of dataflow, selected model and intended task. However, without any inputs beyond the properties of the ML model itself, an ML algorithm can do nothing but blindly guess at transformation parameters. Hence, the simplest extension is to allow the IDC to pass ‘instructional data’ to the parameter controller, as demonstrated by step 8 from the LSE example. Instructional data can be thought of as training data, but also crucially includes data used to evolve a model during adaptation, an important concept explored within Section 9. Notably, not all ML algorithms need instructional data, as the GA example in Fig. 3 demonstrates. However, to constitute a learning process, random parameter guesses must eventually be improved by some form of data-driven feedback, whether immediately calculated by an accuracy evaluator or supplied by a delayed evaluation downstream. This is why the Fig. 4 schematic provides two sources for learning-driven accuracy evaluations.

Once mechanisms for instructional dataflow and accuracy evaluations are fully incorporated, the ML component in Fig. 4 becomes capable of encompassing an extensive variety of ML scenarios. These include the following types of learning:

- Unsupervised Learning – An ML algorithm generates an ML model purely from instructional data, without being provided any external guidance in the form of an expected $Q \rightarrow \mathcal{R}$ mapping. The ML model is thus completely subject to the internal biases of the ML algorithm. Although there are methods to quantify model quality with respect to the intent behind certain algorithms, there is no intrinsic accuracy to evaluate, i.e. there is no ‘ground truth’.
- Supervised Learning – ML-model inputs arrive packaged with expected outputs, i.e. labels or target signals. An ML algorithm can learn from these directly, as instructional data, and/or indirectly, via accuracy evaluations that compare model outputs with expected outputs.
- Reinforcement Learning (RL) – Expected $Q \rightarrow \mathcal{R}$ mappings are not available, but the quality of different parameter selections are still comparable, with an indirect ‘reward’ metric guiding improvement. Some RL strategies can derive this heuristic from instructional data, but, more commonly, model outputs must be assessed downstream with the evaluation then returned to the algorithm.

Hybrids of these scenarios are also represented, such as semi-supervised learning, for which training data arrives as a mixture of labelled and unlabelled instances. Semi-supervised learning typically leverages statistical information from the latter to improve the predictive knowledge afforded by the former. In all cases, learning procedures are mostly internalised by the parameter controller, if not entirely, no matter if they formulaically compute ML-model parameters, e.g. the LSE, or do so in iterative fashion, e.g. k-means clustering.

As a side note, it is worth highlighting that the representational generality provided by the Fig. 4 schematic does mean that instructional data and transformation-input data can often occupy different vector spaces. This is obvious in most forms of supervised learning, where ML algorithms train on $Q \times \mathcal{R}$ while ML models operate in Q -space. Unsupervised learning algorithms are much more varied. While k-means clustering may both train and deploy on Q , the Apriori algorithm used in association learning is a radically contrasting example; the ML algorithm operates on sets of objects, while the ML model operates on queries depicting implication rules. The two do not inherently embed within the same vector space. Consequently, within the Fig. 4 representation, the

interface-supported IDC must be capable of identifying and redirecting heterogeneously structured data into the appropriate channels.

At this point, there are a few miscellaneous comments to add about the ML component depicted in Fig. 4. First of all, just as accuracy evaluations can be received from downstream, they can also be augmented and propagated upstream, hence the inclusion of the ‘propagated accuracy evaluation’ arrow in the schematic. This enables representation of automatic differentiation techniques, such as backpropagation, but these cannot be discussed until considering multiple components, done in Section 4. Secondly, the ability to keep multiple ML-model instances within memory via the transformation set implies that, with appropriate upgrades for the ODC, homogeneous ensembling could be implemented internally within an ML component. Discussion on ensembles in general is delayed until Section 8. Finally, closer inspection of the GA example in Fig. 3 suggests that the deletion/construction of MLP objects in steps 16–17 is superfluous when simply changing the parameter values of the existing MLP population would be sufficient. However, in so doing, the example demonstrates via steps 18–19 how an ML component could communicate with an external resource manager, distributing new model instances across available hardware. Resource management in the ML context is discussed in Sec. 11.

In summary, the schematic in Fig. 4 broadly captures standard ML operations, including learning that is unsupervised, supervised, and reinforcement-based. Within this context, automation of the parameter controller, i.e. model training/development, is essentially what spawned the field of ML. However, it is important to note that the same ML model can be tuned by many different ML algorithms. For instance, a linear function may be tuned via formulaic parameter estimation, e.g. with the LSE or Theil–Sen regression, or via an iterative method, such as is typically done for training a perceptron. Additionally, with so many forms of data transformation to choose from, the question arises: given a particular ML task, which ML model/algorithm should a data scientist use? The attempt to answer this question in a systematic manner is the foundation of AutoML.

3 ALGORITHM SELECTION AND HYPERPARAMETER OPTIMISATION

The modern use of the abbreviation ‘AutoML’ to represent the field of automated machine learning arguably stems from a 2014 International Conference on Machine Learning (ICML) workshop. Specifically, while automating high-level ML operations has broad scope and thus an extensive history, as will be discussed in other sections, it was around this time that, fuelled by a series of advances in optimisation strategies, it became not just feasible but also effective to leave the selection of an ML model/algorithm up to a computer. Accordingly, with the optimisation community seeding the modern narrative of automation in ML, this is a logical starting point when discussing the evolution of standard ML into AutoML.

First, though, a caveat: while we have made an explicit distinction between an ML model and an ML algorithm, or a parameter controller and data transformation in Fig. 4, scientific literature in the field of ML can be loose with terminology [316]. Given how closely models are coupled with their training algorithms, e.g. ‘support vector machine’ (SVM) often referring to both, sections of the ML community use ‘algorithm selection’ and ‘model selection’ interchangeably. In practice, model/algorithm selection refers to a data scientist or high-level system swapping out an entire ML component for another, seeking a pairing of ML model and ML algorithm that is optimal for the task at hand.

Semantics acknowledged, the algorithm-selection problem is far older than the current wave of AutoML research, with its conception often attributed to the 1970s [286]. It was originally posed in fairly abstract terms, seeking a general framework that, given a set of problems and a set of problem-solving algorithms, could identify an optimal algorithm for each problem, according to some specified measures of performance. However, at the time and within the ML context, there was

no feasible procedure to both systematically and efficiently search such an arbitrarily large space. After all, applying an ML algorithm and training a single ML model could already take significant time, especially on the hardware of that era. Nonetheless, with theory and technology progressing over the decades, research attention turned to a limited form of model selection: hyperparameter optimisation (HPO) [68, 373].

Whereas a parameter of an ML model is a degree-of-freedom tuned by an ML algorithm during model development, a hyperparameter is a characteristic of either the model or algorithm that is set to a fixed value at initialisation. For example, with respect to a typical DNN trained via backpropagation and stochastic gradient descent (SGD), neural connection weights are parameters, while network size and the SGD learning rate are both hyperparameters. As a side note, distinctions have occasionally been drawn between model-based and algorithm-based hyperparameters, e.g. the latter being called ‘training tunables’ [74]. Whatever the case, it is quickly evident that hyperparameters already define an extensive space of variation within a fixed class of models/algorithms. Systematically comparing two distinct types of model, like an SVM against an MLP, adds yet another layer of challenge. It is for this reason that, right up until the dawn of the modern AutoML era, researchers have sometimes used ‘model selection’ in a more limited sense, i.e. strictly in reference to tuning hyperparameters for fixed model types [59, 269].

Critically, hyperparameters can have a major effect on the suitability of a data transformation for any particular dataset. This would seem obvious, and ML novices are often taught early on to explore model performance under hyperparametric variation, yet this lesson remains commonly enough ignored to justify warnings against the use of default hyperparameter values [20]. Emphasising the surprising novelty of this point, recent publications in the field of software defect prediction have likewise announced that default classifier settings can perform poorly [280, 333, 334]. In fairness, identifying that hyperparameters can define a poor model/algorithm is far easier than finding good ones; this is the goal of HPO.

Typically, HPO is expressed as a minimisation problem for a loss function, L , as calculated for an ML model/algorithm, A , that is applied to sets of training and validation data, D_{train} and D_{valid} , respectively. Model/algorithm A is itself dependent on a set of hyperparameter values drawn from a structured space, i.e. $\lambda \in \Lambda$. Some form of cross-validation is also often involved, drawing D_{train} and D_{valid} k times from a larger dataset to statistically average loss. With this notation, HPO is written as

$$\lambda^* \in \operatorname{argmin}_{\lambda \in \Lambda} \frac{1}{k} \sum_{i=1}^k L(A_\lambda, D_{\text{train}}^{(i)}, D_{\text{valid}}^{(i)}). \quad (1)$$

The most standard approach to this problem, especially when manually applied, is a grid search through hyperparameter space. Another relatively simple alternative is random search, which tends to find more accurate models much more efficiently [31]. Some packages, e.g. H2O AutoML [146], are content to use grid/random search for model selection on the balance of performance versus complexity.

The development of HPO algorithms that incorporate exploitation, not just exploration, has historically been quite gradual. Many attempts would restrict themselves to hyperparameters with continuity and differentiability conditions; these include gradient-based methods [28] and bi-level optimisations over combined parameter/hyperparameter spaces [29]. However, among HPO efforts, Bayesian-based approaches began to appear particularly promising, often assessed against frequentist methods due to the overlap between ML and the field of statistics [144, 182].

Bayesian optimisation (BO) [314] is usually considered to have been birthed in the 1970s [255], although foundational concepts were explored much earlier. A core idea behind the approach, introduced at least a decade earlier [214], is to use stochastic processes to approximate unknown

functions. These approximators are sometimes called surrogate models or response curves [180]. Plenty of literature has been dedicated to analysing surrogates and their utility [88, 165, 168], although Gaussian processes remain a traditional standard. Regardless, once a surrogate has been chosen, BO uses this approximation to drive iterative evaluations of the unknown target function, with each exploration coincidentally tightening the fit of the surrogate. Technically, each evaluation is advised by an ‘acquisition function’ based on the surrogate, which estimates the benefit of evaluating any point along the unknown function. A commonly used metric for this is ‘expected improvement’, although alternatives have been proposed, such as ‘entropy search’ [156], which focusses on maximising the information gained about an optimum rather than simply moving closer towards it. More in-depth details about BO can be found in tutorials elsewhere [48].

In the meantime, in parallel to BO gradually developing as a feasible approach for standard HPO, the ML community began to return to the question of full model/algorithm selection, with the 2006 Neural Information Processing Systems (NIPS) conference organising an early challenge dedicated to this topic [145]. As a result, possibly the first method for full model search was published not long after [99]. It was based on particle swarm optimisation (PSO) and even tackled the more ambitious problem of multi-component ML models, discussed later in Section 4.

However, from a model-search perspective, the genesis of the modern AutoML era is often synonymous with crucial advances in BO. Specifically, while some HPO strategies could handle issues caused by mixing continuous and categorical hyperparameters, conditional values were more challenging. For instance, if one hyperparameter denotes the kernel that an SVM employs, a value of ‘polynomial’ requires a polynomial-degree hyperparameter to also be chosen. Similarly, a ‘radial basis function’ value opens up the use of a Gaussian-width hyperparameter. In effect, the hyperparameter space for many models/algorithms resembles a complex hierarchical tree. Nonetheless, once these conditional hyperparameters could properly be treated [169], a new method for generic HPO was published, namely Sequential Model-based Algorithm Configuration (SMAC) [167], which uses random forests as surrogates for unknown loss functions that operate on hierarchical domains. Incidentally, the SMAC publication also popularised the phrase ‘sequential model-based optimisation’ (SMBO) to describe the Bayesian-based procedure. Two search methods produced soon after, namely the Tree-structured Parzen Estimator (TPE) approach [34] and Spearmint [318], are both considered to be popular SMBOs, although the latter was notably not designed to handle conditional parameters.

Full model search was quickly identified as a natural extension of HPO. In 2013, the combined algorithm selection and hyperparameter optimisation problem (CASH) was formalised [338], written as

$$A_{\lambda^*}^* \in \operatorname{argmin}_{A^{(j)} \in \mathcal{A}, \lambda \in \Lambda^{(j)}} \frac{1}{k} \sum_{i=1}^k L(A_{\lambda}^{(j)}, D_{\text{train}}^{(i)}, D_{\text{valid}}^{(i)}), \quad (2)$$

where the optimisation is applied across all ML models/algorithms of interest, $A^{(j)} \in \mathcal{A}$, and their associated hyperparameter spaces, $\Lambda^{(j)}$. However, the important insight bundled into CASH was that combinations of ML model and ML algorithm could be described as just another categorical hyperparameter at the root of a hierarchical tree [35, 338]. This meant that generic SMBO algorithms like SMAC and TPE could be applied directly to varying model/algorithm types. Accordingly, 2013 marks the release of the first AutoML package based on SMBO, namely Auto-WEKA [338].

Crucially, Auto-WEKA popularised a form of modularity that has essentially been ubiquitous across AutoML packages, wherein the mechanism for solving HPO/CASH acts as a high-level wrapper around low-level ML libraries, these often being produced by third-party developers. Indeed, as the name implies, Auto-WEKA applies its selection method to a pool of classification

algorithms implemented by the WEKA package. In effect, or at least if implemented well, the optimisation routine can be abstracted away from ML-model particulars.

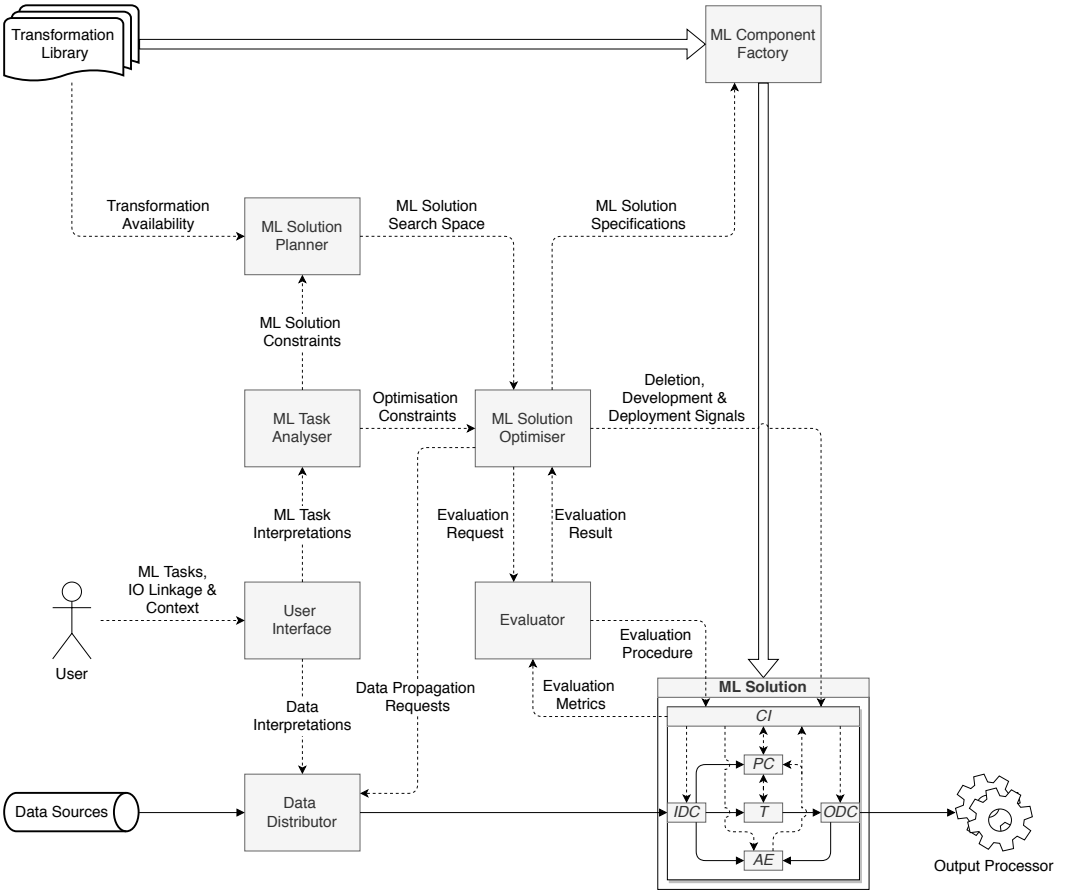


Fig. 5. A high-level schematic of a minimalist AutoML system, demonstrating how a CASH-solver can be incorporated in place of manual model selection within an ML platform. An interface allows a user to describe a desired ML task, along with inputs and outputs to the system. Once the task is interpreted, with relevant constraints passed onwards, a solution planner advises a CASH-solving optimiser in terms of transformation availability and associated ML-solution search space. The optimiser proceeds to iteratively select candidates for an ML solution, instructing a factory to instantiate relevant ML components. An evaluator serves to assess these candidates, while a data distributor feeds training/validation data to the ML components during assessment. The distributor also propagates queries once optimisation concludes and an ML solution is deployed. Every ML component contains a data transformation (T), a parameter controller (PC), a component interface (CI), controllers for inflow/outflow data (IDC/ODC), and an accuracy evaluator (AE). Dashed arrows depict control and feedback signals. Solid arrows depict dataflow channels. Block arrows depict the transfer of ML components.

For illustrative purposes, a variant of this design pattern is shown in Fig. 5, applying model selection via optimisation to a library of transformations and their associated training algorithms. In this example, a user interface (UI) for the AutoML system fields user requests, such as the desire to run a classification task. Abstractly put, the user must also link the system to input/output

(IO) and ensure that this IO is sufficiently described, e.g. where the data sources are, what their formats are, where model responses should be sent, and so on. This provided context allows developmental/query data to be propagated as requested by a data-distributor module.

Central to Fig. 5, and any modern AutoML system, is the ML-solution optimiser, e.g. an implementation of SMAC. Systems vary in how much flexibility a CASH-solver is afforded, so the schematic includes an ML-task analyser that passes optimisation constraints onwards, e.g. time allowed per iteration. Often, this analysis can also provide some hard constraints on the ML solution to be expected, e.g. excluding regression-based models for a classification task. These constraints must be examined in the light of which data transformations are available to the AutoML system, so an ML-solution planning module is tasked with actually detailing the search space to the optimiser.

From here, the encapsulation of ML models/algorithms with their low-level control/management mechanisms, detailed in Section 2, makes HPO convenient to depict. During optimisation, the CASH-solver iteratively directs a factory to instantiate ML components according to multi-dimensional hyperparameter guesses, with types of ML model/algorithm included in the selection process. Each ML component becomes a candidate ML solution, with the optimiser deciding when to delete it. During each iteration of optimisation, a high-level evaluator is tasked with assessing the candidate; it directs the ML component via interface to undertake an evaluation procedure, using data that is simultaneously sent to its IDC. As Section 2 implied, this is where it seems most logical to automate cross-validatory mechanisms. Finally, when the optimiser hits its stopping criteria, it ensures that the optimal ML component it found is constructed and then signals it to train on a complete set of developmental data, requested from the data distributor. When this is finished, it signals the ML solution to deploy, requesting the data distributor to pass on any incoming queries, and the ODC of the ML component subsequently outputs all responses.

As before, we do not claim that this is necessarily the best way to partition AutoML processes into modules and network them up, particularly from an implementational perspective. Deeper commentary on the challenges of integration is provided in Section 14. Moreover, bespoke frameworks arguably do not need this level of complexity, e.g. a restrictive transformation library may not require a planner to constrain search space. However, if the field of AutoML is dedicated towards automating an increasing spectrum of user-desired ML operations, all concepts depicted in Fig. 5 must be integrated in some fashion.

Returning to SMBO methods in general, research and development continued in the years following the release of SMAC [240]. One of the earliest aims of the sub-field was to decide whether some SMBO strategies could find optimal models faster and more accurately than others. As Eqs (1) and (2) show, this is not a trivial question; a loss function is critically dependent on the data used to train and validate a model. Thus, Python-based benchmarking library HPOlib was proposed to standardise this [95]. It was used in conjunction with several search strategies built around SMBO package Hyperopt [33], which only implements TPE by default, to make preliminary assessments [32], although these served more as a proof of concept than strong recommendations for any one SMBO method. Subsequent attempts by the ML community have attempted to mitigate the resource costs of benchmarking [96], discuss ways of ranking the SMBO methods [79], and assess strategies within specific contexts such as multi object tracking [243]. Of course, Hyperopt is only an early entry among several libraries that provide SMBO implementations. For example, an R-based machine-learning toolbox for model-based optimisation (mlrMBO) extends into multi-objective optimisation [40], while Python-based Sherpa prioritises parallel computation [157].

Other non-Bayesian strategies have also shown promise for both HPO and CASH. Evolutionary algorithms such as PSO and GA have been heavily investigated [55, 99, 126], shown to perform better than gradient-based methods in certain nonlinear contexts [81], although their advantages are most discernible for multi-component ML models, discussed in Section 4. A lot of recent focus

has also gone into reinforcement-based strategies. Although RL becomes particularly pertinent for neural network design [21, 386], discussed in Section 5, the long-studied multi-armed bandit (MAB) problem [309, 337] has been linked in general fashion to HPO. This has led to a number of MAB-based solvers being proposed, e.g. the multi-armed simultaneous selection of algorithm and its hyperparameters (MASSAH) method [94] and the more recent Hyperband [230]. Accommodating the HPO setting does often require certain MAB assumptions to be tweaked, such as treating the performance ‘reward’ of selecting an ‘arm’, i.e. model, as non-stochastic [174]. Other research has continued relaxing assumptions, such as exploring nonstationary contexts where the performance reward may change over time [250]; this is perhaps relevant to the selection of ML models/algorithms in the context of adaptation, discussed in Section 9. Notably, attempts have also been made to fuse Bayesian theory with MAB strategies, such as using correlations between arms to advise subsequent arm-pulls [159], and one such culmination of these efforts is the BO-HB method [101], which aims to balance the exploitation of Bayesian optimisation with the exploration of Hyperband. Then there are even more exotic variants for CASH, such as combining RL with Monte Carlo tree search (MCTS) [86].

Regardless of CASH-solver chosen, it is commonly accepted that searching for a task-optimal ML component, i.e. model/algorithm, is extremely computationally expensive, likely to take hours if not days on modern hardware. Accordingly, it has been suggested that, even in the more limited case of HPO, where research once prioritised over-fitting avoidance, the big-data era has shifted focus to search efficiency [242]. Meta-learning, discussed in Section 7, has been suggested as one possible way to boost the selection of ML models/algorithms, such as pruning hyperparameter space based on previous experience [362]. However, in the absence of external information, other speed-up mechanisms have been sought.

Some researchers have suggested that the easiest way to reduce search space is to simply design useful ML models/algorithms with fewer hyperparameters [171]. Even decomposing CASH once more into smaller independent subproblems, as is done via the alternating direction method of multipliers (ADMM), has had appeal [235]. Others have focussed on the hyperparameters themselves, assessing their impact on ML models/algorithms either in general [278] or in specific contexts, such as with random forests [279]. The hope is that identifying ‘tunable’ hyperparameters, i.e. ones that model performance is particularly sensitive to, will allow other settings to be ignored, constraining search space. This is also the motivation behind several analysis of variance (ANOVA) studies [166, 370]. However, all such investigations acknowledge the choice of training datasets as complicating factors, making it difficult to generalise tunability results.

Complementary to constraining search space is cutting off unpromising forays via early termination. In simplest form, this is abandoning a search if the performance of an evaluated ML model/algorithm is not bettered after a subsequent number of selections [370]. A more involved approach for CASH-solvers that parallelise evaluations is to initially test a large sample of hyperparameter configurations, maximising exploration, and then gradually decrease the size of subsequent samplings, thus honing in on the best-performing models [101, 230]. This is often bundled with a resource allocation strategy, e.g. of training time, so that a fixed budget is spread less thinly across fewer candidate models when they are expected to be better performing.

Engineering search-boosts can also target lower-level aspects, such as the training/validation time it takes to evaluate a single candidate model. For instance, in the 1990s, Hoeffding races were proposed as one way to discard poorly performing models quickly during cross-validation. Specifically, confidence bounds on model accuracy would be updated per iteration of training/validation, with any model being immediately eliminated from contention if its maximum bound fell below the minimum value of any competitor [247]. Feedback-driven ML algorithms, such as backpropagation-based SGD, are similarly open to interruptions prior to completion; learning curve

extrapolations have been explored as quick estimates of final model performance [82], allowing for early termination.

Naturally, the logical extreme of speeding up training/validation is to constrict the data itself. Even a small subsample can indicate the expected performance of a fully-trained model [275], especially if the sample is density-preserving and reflects the characteristics of the full dataset [50–52] or involves some bootstrapping technique [12, 13]. Hence, similar to a graduated allocation of training time, some HPO strategies use an increasing amount of training data for sequential model evaluations, honing in on better performing regions of hyperparameter space after cheap exploratory estimates [357]. Sometimes, for CASH, this is coupled with other iterative updates, such as progressively decreasing the pool of ML model/algorithms to explore [240, 381].

Further efficiency upgrades may be possible depending on CASH-solver specifics. For instance, the acquisition function used by SMBO methods is a prime target for augmentation. Metrics such as ‘expected improvement per second’ and similar variants have been shown to guide BO away from regions of hyperparameter space where evaluations are estimated to be time-expensive [317]. A more recent example involving a method called Fast Bayesian Optimisation of Machine Learning Hyperparameters on Large Datasets (FABOLAS) gives itself the freedom to select how much of a dataset a model should be trained on; the subsample size is factored into its entropy search metric, forcing BO to balance the desire for more instructional data against the desire for cheap model training [202]. Notably, although the idea veers towards meta-learning principles discussed in Section 7, entropy search has also previously been boosted by factoring correlations between related tasks [331].

On a final practical note, upon appreciating that the field of HPO is constantly evolving, developers intent on designing AutoML systems may ask; what CASH-solver should I plug into my codebase according to the state of knowledge as of 2020? According to recently published comprehensive benchmarking, the state-of-the-art answer is BO-HB, assuming well-behaved datasets where random subsamples are representative of the whole [373]. Alternatively, for ML tasks with data sources that do not obey those statistics, the same survey advises BO strategies for small hyperparameter spaces and PSO for large ones. These suggestions appear to be a good rule of thumb, although the sparsity and limitations of benchmarking studies still preclude any encompassing conclusions.

In summary, AutoML, as it has been popularised, is inextricably linked to the topic of optimisation. Whereas standard ML and its search for model parameters can be encapsulated within an ML component, AutoML has sought ways to automate the selection of an ML component itself. Accordingly, ML software is rarely considered an AutoML package unless it has implemented some form of CASH-solver at a high level, and ongoing research continues to seek efficiency gains in this area. However, in under a decade, AutoML has come to represent ambitions far beyond CASH. A recent publication has even pointedly suggested that HPO may not be necessary for high-performance AutoML [98]. In effect, the scope of the field is both fluid and expansive, encompassing anything that may benefit the desires of an ML practitioner; this is not limited to maximising ML-model accuracy and minimising runtime. For instance, some users and ML tasks may require ML models/algorithms with significant complexity. In these cases, one ML component may no longer be an effective representation for an entire predictive system.

4 MULTI-COMPONENT PIPELINES

The purpose of ML, as expressed in Section 2, is to algorithmically ingest data in some fashion to inform the production of a maximally desirable mapping from a query space to a response space, i.e. $M : Q \rightarrow \mathcal{R}$. While we have thus far implicitly treated M as a simple singular object, the reality is that ML tasks can involve substantially complicated ground-truth functions; users

may need to employ correspondingly complex ML models to have any reasonable chance at a good approximation. The problem with this is that arbitrarily general nonlinear mappings are intractable. Complex ML tasks can only be attacked by chaining together simpler data transformations that are individually well understood and better behaving, if not outright linear. Of course, it is still a significant challenge to form an ‘ML pipeline’ of well-tuned components that, when concatenated, acts as a good ML model for a task at hand. So, with AutoML successful in tackling standard CASH, the natural extension is to automate the construction of a multi-component predictive system (MCPS) [300, 303].

However, as before, it is worth highlighting semantic ambiguities first. An example of a standard three-component ML pipeline is one that transforms data defined in a query space, pre-cleaned for convenience, through two intermediate ‘feature’ spaces and into the response space, i.e. $P_{1+2+3} : Q \rightarrow \mathcal{F}_1 \rightarrow \mathcal{F}_2 \rightarrow \mathcal{R}$. Some data scientists would consider the final segment of this ML pipeline, sometimes called a predictor, to be an ML model, e.g. an evaluable classifier/regressor, while the previous two segments would be called pre-processors, e.g. feature engineers. Deep-learning specialists may alternatively call the entire ML pipeline an ML model, in recognition of feature-generating layers being internal to a convolutional neural network (CNN), as well as the fact that a singular backpropagation-based algorithm typically trains all of its layers. We avoid this debate; the abstract language of ‘ML components’, ‘data transformations’ and ‘parameter controllers’ enables a conceptual AutoML architecture to represent an encompassing range of ML scenarios.

Indeed, given the ML-component wrapper introduced in Section 2, it is relatively simple to fashion a higher-level analogue. Specifically, Fig. 6 demonstrates how the ML solution within Fig. 5 can be upgraded into ML-pipeline format. All associated modules in the high-level schematic can now be presumed to fashion entire pipelines, as opposed to solitary ML components; these include the planner, optimiser, and factory. As for the suggested design of an ML pipeline, we recommend similar elements of wrapper-based control to those provided by the ML component. First of all, the ML pipeline should contain an interface that relays communications between external modules and individual components. Optimising an ML pipeline becomes a complex balancing task with respect to the optimisation of individual ML components, and advanced HPO strategies catering to an MCPS must be actionable. This means individual ML-component evaluations, if available, should be disseminated to external modules as well. On that note, while the predictive accuracy of ML pipelines will typically mirror the accuracy of their final component, this is not always the case, particularly if the ML-pipeline output is an aggregation involving multiple ML components. Essentially, installing an accuracy evaluator at the pipeline level allows an ML pipeline to be assessed independently of any ML-component evaluations. Finally, pipeline-specific controllers for inflow/outflow data are also advised, with the pipeline IDC directing dataflow to all component IDCs. This allows expected outputs to be shuttled to wherever an accuracy evaluator is in play, while also allowing each individual ML component to be fine-tuned, assuming that training data can be provided within its relevant feature space.

In principle, the ML-pipeline paradigm enables arbitrary complexity. An example is provided in Fig. 7, with most aspects of pipeline control omitted to avoid clutter. Here, each incoming instance of data, defined in Q -space, is a concatenation of an image in Q_1 -space and tabular data in Q_2 -space. With ML-component IDCs acting as gatekeepers, possibly declaring their permissible inflow spaces to the ML-pipeline IDC via indirect means, the data is essentially partitioned between parallel tracks. Each image is passed through a CNN of three layers, i.e. convolutional (CL), max-pooling (MPL) and output (OL), so as to be classified in a binary response space, \mathcal{R} . This net transformation passes through two intermediate feature spaces, \mathcal{F}_1 and \mathcal{F}_2 . The tabular data instead takes the alternate track, being refined by principal component analysis (PCA) and mapped into feature space \mathcal{F}_3 . A decision tree (DT) makes an early attempt at classification, $\mathcal{F}_3 \rightarrow \mathcal{R}$, while k-means

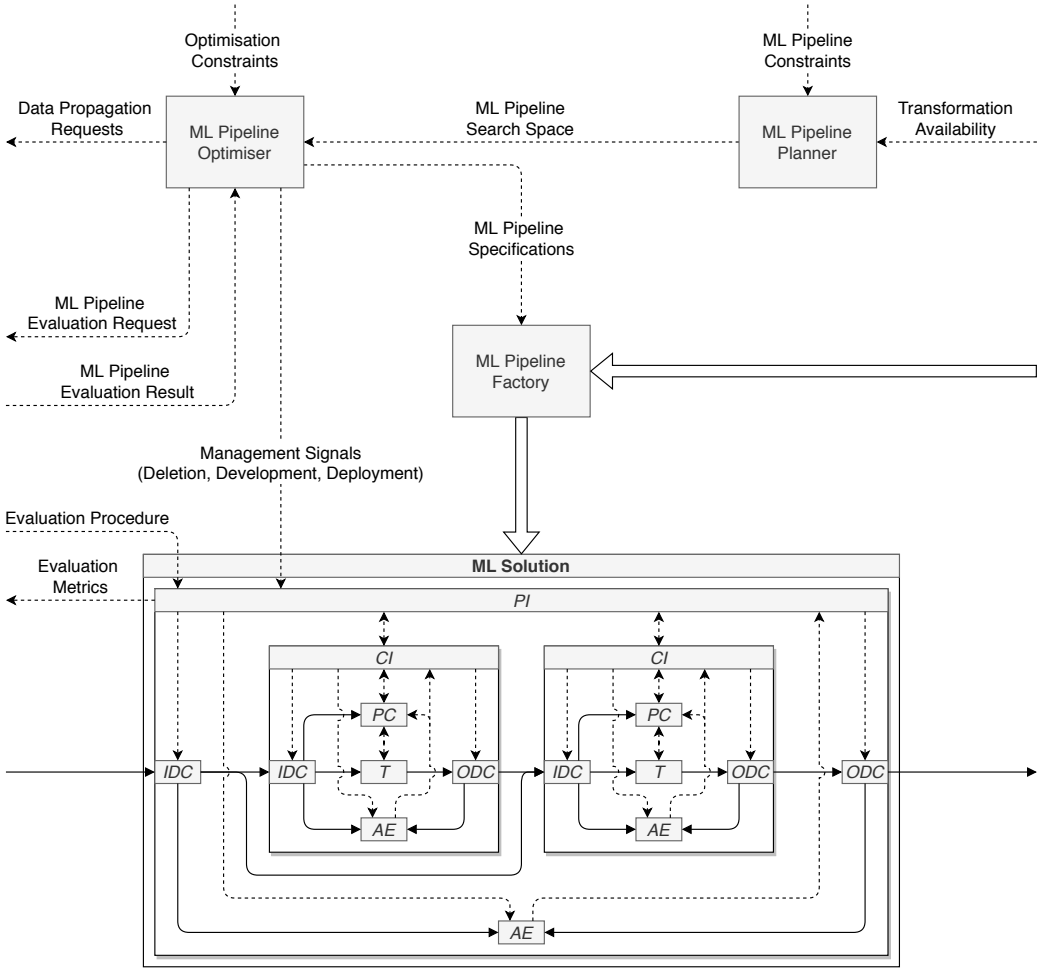


Fig. 6. A schematic of a two-component ML pipeline acting as an ML solution within an AutoML system, with immediate modular environment pulled/renamed from Fig. 5. A pipeline interface (PI) relays communications between external systems and each component interface (CI), which in turn communicates with the parameter controller (PC) for each data transformation (T). The PI also manages a pipeline-specific inflow-data controller (IDC), which supplies all the IDCs of individual components, and likewise manages a pipeline-specific outflow-data controller (ODC), which receives output from the tail end of the ML-component arrangement. The ML pipeline also maintains its own accuracy evaluator (AE), mimicking the control structure of an ML component. Dashed arrows depict control and feedback signals. Solid arrows depict dataflow channels. Block arrows depict the transfer of ML components/pipelines.

clustering tries to generate a new feature in \mathcal{F}_4 -space. These outputs are concatenated with the original refined data into a new feature space, $\mathcal{F}_5 = \mathcal{F}_3 \times \mathcal{F}_4 \times \mathcal{R}$; this data is then classified by an SVM, mapping \mathcal{F}_5 back to \mathcal{R} . A final averaging across the outputs of parallel tracks, i.e. the image classifier and the tabular-data classifier, results in a final response.

Importantly, this example demonstrates how an ML solution can involve a mix of strategies. For instance, neural networks can be part of an ML-pipeline search, elaborated in Section 5, if error

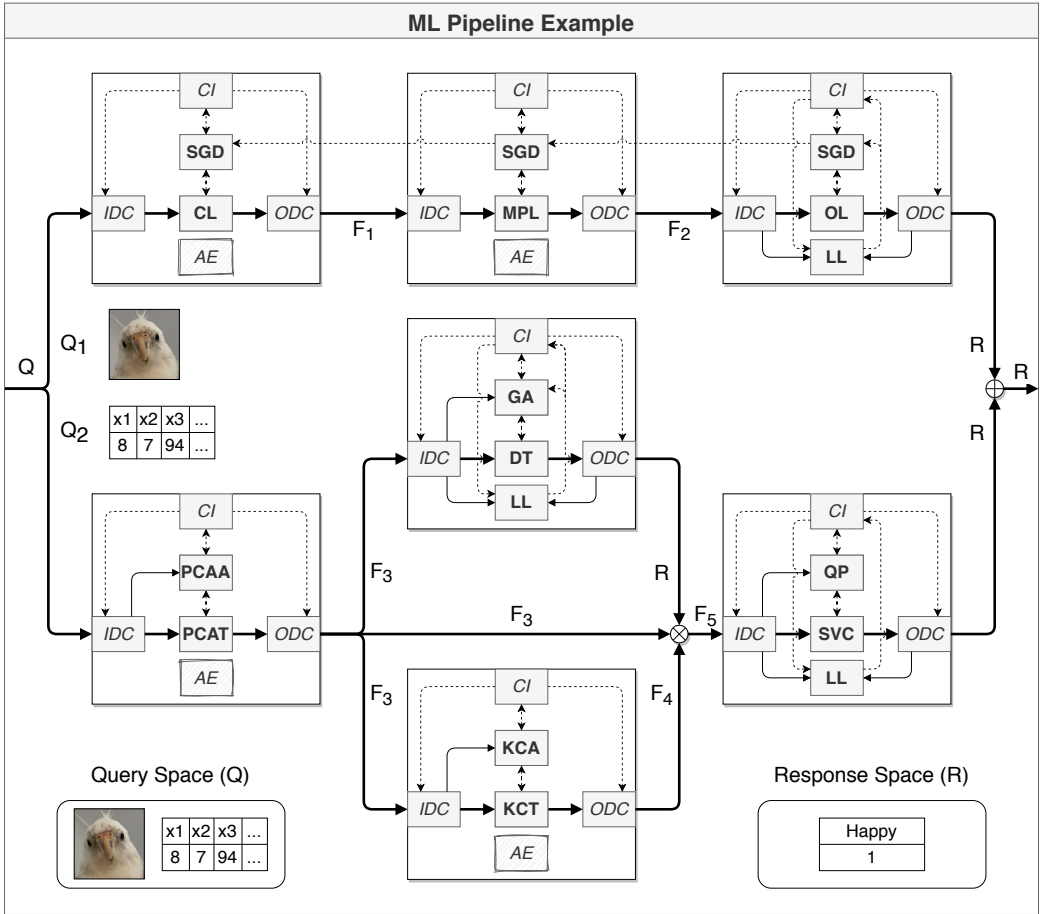


Fig. 7. An example of a complex seven-component ML pipeline, mapping queries of mixed image/tabular to a binary classification response. Each ML component is an instantiation of Fig. 4 in compressed representation, containing a component interface (CI), controllers for inflow/outflow data (IDC/ODC), and an optional accuracy evaluator (AE). Other ML-component abbreviations denote a convolutional layer (CL), max-pooling layer (MPL), output layer (OL), stochastic gradient descent (SGD), logistic loss (LL) calculator, principal-component-analysis transformation/algorithm (PCAT/PCAA), k-means-clustering transformation/algorithm (KCT/KCA), decision tree (DT), genetic algorithm (GA), support vector classifier (SVC), and quadratic programming (QP). Solid arrows depict dataflow channels; thick arrows denote dataflow during deployment. Dashed arrows depict control and feedback signals. Labels along component-external dataflow channels define the spaces that data inhabits at those points; they are detailed in Section 4. The ‘ \times junction’ concatenates data from different spaces. The ‘ $+$ junction’ averages data within the same space. Pipeline-level control elements, other than query-based dataflow, are omitted from the diagram for clarity.

evaluations are allowed to propagate between ML components, as hinted within Section 2. Simple versions of ensembling strategies like ‘stacking’ are also possible on this level [64], as shown with the SVM operating on DT outputs. Moreover, the example represents both algorithms that operate solely on instructional data and those that tune parameters based on evaluation feedback, e.g. the quadratically-programmed support-vector-classifier and the CNN trained against logistic loss, respectively. In contrast, the GA-driven DT can be a fusion of both approaches, using instructional

data to build up trees and evolutionary tactics advised by accuracy evaluations to prune them. As for PCA and clustering, they re-emphasise that unsupervised techniques can be combined with supervised learning. Again, with solid design and implementation on the base level, an ML pipeline can be endlessly inclusive, whether or not the complexity is justified. At the very least, shunting pre-processing operations into the ML pipeline enables AutoML to automate feature engineering [345], discussed further in Section 6.

Turning to a historical perspective, automatically selecting a good ML pipeline for an ML task is not a new idea. Several recommender systems were developed in the 1990s and 2000s that suggested streams of data-transforming processes; these were often based on meta-learning, discussed in Section 7. Likewise, prior to CASH being formalised, there was at least one proposal for an MCPS-based ML architecture [183].

It is not unexpected then that full-model-search pioneers tackled ML pipelines from the start. For instance, one PSO-based attempt optimised over a hyperparameter space that encoded whether or not to use individual pre-processors [99]; this strategy would later be tweaked to divide HPO and ML-pipeline selection between the PSO and a GA, respectively [326, 327]. Similarly, MLbase, a contemporary of Auto-WEKA, focussed on constructing and optimising ‘learning plans’ from series of data operators [211]. Oddly, given that tree-based SMAC can handle ML pipelines natively [106, 107], Auto-WEKA seems almost anomalous in limiting pre-processing to feature selection [208, 338], but, in fairness, its development may have prioritised ensembling instead, a topic discussed further in Section 8.

Since those earlier years, AutoML research has definitively expanded from HPO through CASH to ML-pipeline optimisation [45]. Some coding packages specifically promote these features, such as AutoWeka4MCPS [298, 301], built directly on top of Auto-WEKA. Notably, appreciating that complex pipelines can be challenging to interpret for users, AutoWeka4MCPS applies Petri net representation [346] to a constructed MCPS for the sake of transparency [302]. Then there are AutoML systems that fully embrace ML pipelines with theoretically arbitrary complexity, such as the GA-driven Tree-based Pipeline Optimization Tool (TPOT) [267], which has further explored the parallelisation of feature generation/selection as part of a prostate-cancer case study [268]. The appeal of such an inclusive approach, as indicated by the Fig. 7 example, comes from the fact that relaxing ML pipelines from linear chains to general directed acyclic graphs (DAGs) increases their representative potential. Attempts to generalise backpropagation for arbitrary ML pipelines have similarly supported DAGs [254].

Nonetheless, designing an MCPS has many challenges. Chief among them is ensuring ML components can be validly combined together into an ML pipeline. This is less of a problem when data is clean, numerical and vectorised. Messy real-world data, on the other hand, remains a challenge for learning systems, both adaptive [383] and otherwise. Indeed, the Challenges in Machine Learning (ChaLearn) 2015-2016 AutoML competition found that all participants but one were unable to deal with sparse data [143, 329]. It is then no surprise that folding missing-value imputation into an ML pipeline requires careful thought, as exemplified by TPOT-based research [124, 125]. Alternatively, accepting that some pipelines will fail and quickly identifying them is another tactic, which is employed by the recently published ‘AVATAR’ method [263, 264]; this approach avoids the computational cost of setting up and executing an entire ML pipeline by evaluating a simpler proxy.

In practice, most AutoML systems avoid ML-pipeline generality from the outset, using hard constraints to avoid both invalid compositions and unnecessarily bloated search spaces; the planning module in Fig. 6 enshrines this approach. For instance, if a data scientist faces an ML task involving image-based inputs, they will probably benefit from using convolutional neural layers. Likewise, dataset normalisation is likely to be a transformation employed early on within an ML pipeline. In

fact, forming ontologies of operators and linking them with useful workflows has been a priority of the data-mining research field even before the current wave of AutoML [199]. Thus, using similar concepts, MLbase provides one such example in which a ‘logical learning plan’ acts as a template for a ‘physical learning plan’, i.e. an instantiated ML pipeline [211]. Likewise, whereas TPOT constrains its pipeline search with a Pareto front that factors in the number of ML components within a solution [267], an alternative GA-driven AutoML framework named ‘REsilient Classification Pipeline Evolution’ (RECIPE) uses a grammar for finer control [77].

Notably, the problem of building optimal ML pipelines within hard constraints shares overlaps with the field of automated planning [178]. Accordingly, the concept of hierarchical task networks (HTNs) has been adopted to systematically describe valid ML pipelines, whether in the design of data-mining tools [199, 200] or more recent AutoML systems like ML-Plan [257, 258]. Researchers behind ML-Plan imply that, due to the recursive nature of HTNs, it is still possible to explore ML pipelines with arbitrarily long pre-processing workflows, matching the generality of TPOT without wasting evaluation time [359].

Whatever the design, the actual optimisation of ML pipelines remains challenging, given that the typical CASH already corresponds to optimising a one-component pipeline. Genetic programming [209] is one particularly favoured technique in this context as it natively works with ordered sequences of operators; it is used in TPOT [267] and RECIPE [77] among other AutoML systems. An asynchronous version of this evolutionary algorithm is implemented by the Genetic Automated Machine learning Assistant (GAMA) [136]. However, SMBOs remain employed as an alternative, with Fast LineAr SearchH (FLASH) as a recent attempt to upgrade Bayesian optimisation for ML pipelines, separating CASH from pipeline search and applying a linear model to error propagation across ML components [382]. A subsampling method called Bag of Little Bootstraps (BLB) has likewise been used to speed up BO-based pipeline search [12, 13]. Naturally, as with CASH, there are also yet other approaches, such as the MCTS used in ML-Plan [258].

However, to date, there is no conclusively best-performing search strategy for ML pipelines. It is not even clear how an MCPS complicates hyperparameter space, with one exploration of fitness landscapes finding frequent disperse optima and situations where basic grid/random search methods are highly competitive [126]. Another investigation supports the multiple-optima finding, with repeated ML-pipeline optimisations producing inconsistent results depending on how the search was initialised [303]. This is not entirely unexpected; the multiplicity of distinct but similarly performing models has attained the nickname of ‘the Rashomon effect’ [47]. Nonetheless, it is still an open question as to how AutoML should select an MCPS when faced with this conundrum, let alone whether exhaustive searches are worth the computational resources when an easily found local optimum is ‘good enough’.

Ultimately, as optimisation strategies for ML pipelines improve, these principles of automation continue to envelop new settings. For instance, the Hyperopt library has been applied to signal processing pipelines [148], while causal impact analysis operators have been considered as extensions to the standard pool of ML components [164]. Finally, a recently published system named AutoML-Zero has pushed atomicity to the extreme, applying genetic programming to simple mathematical operators so as to build up predictors and learning algorithms from scratch; this notably includes a rediscovery of backpropagation driven by gradient descent [282]. Such edge cases can be challenging to fit within an encompassing framework, such as the illustrative one built up in this paper, as, despite the pipelining similarities, AutoML-Zero could arguably be described as a one-component HPO problem, just with an extraordinarily high-dimensional hyperparameter space. All the same, this indicates how the diversity of modern AutoML systems can blur the lines of simple categorisation.

5 NEURAL ARCHITECTURE SEARCH

In the era of DNNs, it was inevitable that the principles of AutoML would be co-opted in automating deep learning. Indeed, the ‘AutoDL’ abbreviation has already been adopted by the most recent ChaLearn competition, AutoDL 2019-2020, to draw a symbolic line between the use of tabular data, i.e. the traditional focus of pioneering AutoML, and data domains that deep learning has excelled in processing [236], such as images, videos, speech, and text. The process of automatically finding optimal DNNs and related neural networks for ML tasks has similarly attained the distinct label of ‘neural architecture search’ (NAS) [97, 154, 285, 361].

Neural networks have a special status amongst ML models, in part due to the ‘expressive power’ they are afforded by the universal approximation theorem. A standard feedforward network (FFN) with unrestricted size and a nonlinear activation function, e.g. the rectified linear unit (ReLU), can fit a continuous function arbitrarily well. Accordingly, there are claims that the performance of several state-of-the-art DNNs comes from their capacity for extremely deep representations [153]. The theoretical impact of layer width upon approximating-potential has also been studied [238]. In essence, connectionist models are powerful enough that a deep-learning specialist need not necessarily interface with adjacent topics.

From a conceptual perspective, however, it is difficult to assess whether NAS is truly a separate category of AutoML, despite often being treated as its own chapter of the story [170]. For one thing, society had only just begun to shift its focus to deep learning [212] during the first wave of AutoML-package releases; it is no surprise that wrappers around ML libraries designed before 2010, e.g. Auto-WEKA [338] and Auto-sklearn [106], treat neural networks in a limited way. In contrast, AutoML systems that focus on NAS profit off of more recently engineered foundations, e.g. Auto-Keras [179] and Auto-PyTorch [252]. The distinction between AutoML and AutoDL thus appears somewhat superficial, even if a DNN, as a self-contained model, typically has a far more complicated hyperparameter space than any standard alternative, such as an SVM. This is true, but it unnecessarily confounds reductionist approaches to ML automation.

It is arguable instead that NAS ultimately collapses to the problem of ML-pipeline search. Indeed, as discussed in Section 4, non-output neural layers can be considered equivalent to standard ML pre-processors. That is why the schematic of the ML component in Fig. 4 explicitly includes error propagation signals; this allows a DNN to be broken apart into the pipeline representation exemplified by Fig. 7. Of course, in fairness, modern neural networks are diverse in nature. Any encompassing AutoML framework requires careful engineering to represent as many scenarios as possible, e.g. handling internal state to cater for recurrent neural networks (RNNs). However, these are mostly matters of implementation.

Notably, while NAS has become a prominent aim of ML research within the last few years, efforts in neural-network selection stretch back at least two decades earlier. Evolutionary algorithms were a popular choice for investigating network design, as shown by a survey from 1999 [375], and other early investigations even explored Bayesian theory for making architectural comparisons [241]. As for codebases, an ML suite named Learn-O-Matic [311] was described in 2012 as using RL-based ‘policy gradients with parameter based exploration’ (PGPE) [312] to optimise the structure of feedforward networks. This system, although contemporaneous with Auto-WEKA, appears to have flown under the radar within the larger ML community. This is not unexpected given the sheer volume of deep-learning publications; it can be arbitrary as to which advances stick out to the community and inform best practice, let alone standard practice. Sure enough, in similar fashion to HPO, both grid search and random search remain default approaches to building neural networks, with the former demonstrated by a mammogram classification study [111].

That all acknowledged, 2017 witnessed several conference papers that caught public attention and have stimulated intensifying interest in NAS. One of these proposed using an RNN ‘controller’ to construct a ‘child’ network, layer after layer, by sequentially recommending values for structural hyperparameters [386], e.g. filter width/height per CNN layer. The RNN controller is trained via RL, driven by performance assessments of the resulting child network when applied to an ML task of choice. In terms of the illustrative framework conceptualised within this paper, specifically the elements shown in Fig. 6, the child network is essentially an ML pipeline, while the RNN controller is equivalent to an ML-pipeline optimiser. This kind of approach, searching for an optimal sequence of building instructions, also inspired the ‘MetaQNN’ strategy [21], which is based on Q-learning applied to Markov decision processes.

Naturally, numerous other search strategies were quickly applied to NAS. Some of them were continuations of long-term neuro-evolutionary research; these were shown to produce DNNs that are highly competitive with other state-of-the-art networks, as judged on typical Canadian Institute For Advanced Research (CIFAR) benchmarks [283, 325]. Alongside the standard principles of sampling high-performance candidates from a population, so as to iteratively breed new generations of neural networks, these mechanisms typically involve encoding networks by some genetic representation, so that mutation operators are capable of both adding/removing layers and modifying structural/training hyperparameters. Complementing GAs and RL, another major class of NAS approaches revolves around gradient optimisation. Differentiable ARchiTecture Search (DARTS) is an archetype of this strategy [234], which eschews discretisation and aims to relax network representation into a continuous space. This allows both connection weights and network architecture to be tuned as part of a single bi-level optimisation problem.

Not unexpectedly, the inevitable alignment of AutoML and NAS also brought BO methods into the fold. Auto-Net [251] is a SMAC-based AutoML package released before the current upsurge in NAS interest. Although, in principle, it is capable of searching for fully-connected FFNs of arbitrary depth, its initial publication limited the number of layers to six for practical reasons. More recently, other groups of researchers from the same institution that published Auto-Net have criticised several prominent NAS methods, both RL-based [386] and GA-based [283], for optimising ML-pipeline structure independently of ML-component hyperparameters [380]. Emphasising that NAS and CASH should be solved simultaneously, Auto-PyTorch is an upgrade of Auto-Net that utilises BO-HB instead of SMAC, while also extending configuration search space to include modern deep-learning techniques and structures [252]. The importance of combined hyperparameter and architecture search (HAS) has gained traction, with a recent publication of an ‘AutoHAS’ approach likewise grappling with how to specify configuration space across all levels, albeit for a gradient descent strategy as opposed to BO [84].

Optimisation approaches aside, the topic of NAS proves most valuable to general AutoML research as a case study of an MCPS taken to extremes. Training a single neural network on non-tabular datasets can be computationally expensive, and this cost can balloon dramatically when reiterated numerous times throughout an exceedingly complex search space. A 2020 survey of modern NAS methods lists the number of GPU days each one took to learn from the CIFAR-10 and ImageNet datasets [285]; values in the hundreds and thousands are not uncommon. Accordingly, as with the HTNs employed for general ML pipelines, described in Section 4, the obvious approach to make NAS manageable is to artificially constrain allowable configurations. A popular practice in recent years is to construct networks with ‘cells’ [361], complex substructures that can be reused multiple times within larger templates. So, for instance, a CNN may consist of a number of cells interleaved with input/output and pooling layers; NAS strategies need to decide how many cells to stack together and what the contents of an individual cell are, but this is a significantly smaller search space than optimising cells independently. Nonetheless, it is an open question whether the

simplicity afforded by cell-based search, alternatively called micro-NAS, is worth the loss of layer diversity afforded by ‘global’ search [163], sometimes called macro-NAS.

Beyond constraining search space, many efficiency-based research threads in NAS relate to establishing intelligent shortcuts in the ML-pipeline search process, such as recycling architectures and sharing trained parameter values between candidate networks [285]. Another example is leveraging functionality-preserving morphisms, implemented by the Auto-Keras package, to iterate effectively through potential networks [179]. Notably, in return, Auto-Keras seems to require a decent initial configuration, which will depend on the ML task at hand. In certain cases, it may be sufficient for a user to suggest this starting point, but other NAS approaches lean heavily into transfer learning to automate this jump-start. For instance, one proposed extension to the original RNN-controller scheme [386] is to include an embedding vector that represents a diversity of ML tasks, such that a fully trained RNN-controller is able to suggest a strong candidate DNN for any new ML task, based simply on which previously encountered ML task this new one is similar to [369]. Then there is few-shot learning, an extreme variant of transfer learning, where a well-performing but generic DNN is sought out to serve as a near-optimal initial configuration for a NAS attempt [109]. Many of these speed-up mechanisms are based on or adjacent to meta-learning; see Section 7.

For now, it is too early to make conclusive remarks about NAS as it relates to AutoML as a whole. While the fine details can vary dramatically, reviewed more broadly/deeply elsewhere [97, 285, 361], NAS appears to be a subset of ML-pipeline search; its elements are able to be represented by the abstractions within Fig. 6. However, the research area is constantly evolving, subject to a lot of attention, and novel network designs are constantly being proposed, with some examples mentioned in Section 13. At some point, with an ongoing trend towards packaging network design for biomimetic neurons [90], the principles of NAS may even be extended to spiking neural networks, although how best to approach such a fusion remains extremely speculative. Regardless, as a nascent field stimulating unbridled exploration, NAS has shorter-term problems to address. For instance, robust evaluations of network topology and other NAS outputs [83] are rare. More troublingly, a recent benchmark comparison of random search with several state-of-the-art NAS strategies has found no competitive distinction, simultaneously challenging the effectiveness of constrained search spaces and weight sharing strategies [379]. This result supports a previous assessment, likewise involving random search, that additionally bemoans a lack of reproducibility within the field in general [231]. These have accompanied similar criticisms about the lack of ablation studies, necessary to identify the novel aspects of NAS strategies that truly advance efficiency/performance, as well as a publication bias that does not highlight important negative results [132]. Hence, while NAS research shows potential in automating ML-pipeline construction, the field remains far from fully mature.

6 AUTOMATED FEATURE ENGINEERING

Real-world data for any ML task is rarely structured in an informative and discriminative manner. One way to face this challenge is to design complex predictors capable of drawing highly nonlinear classification boundaries or regression slopes within these difficult data spaces, e.g. SVMs with exotic kernels. Alternatively, one may seek to nonlinearly morph incoming data until it sits within a feature space that can be mapped to expected outputs in a simple fashion, perhaps even linearly. This latter approach is called feature engineering (FE), and its automation falls under the scope of AutoML research.

In principle, like NAS, automated feature engineering (AutoFE) is related to, if not subsumed by, the topic of ML pipelines examined in Section 4. Any early-stage operation, e.g. outlier handling or one-hot encoding, can be considered as a data transformation deserving of its own ML component.

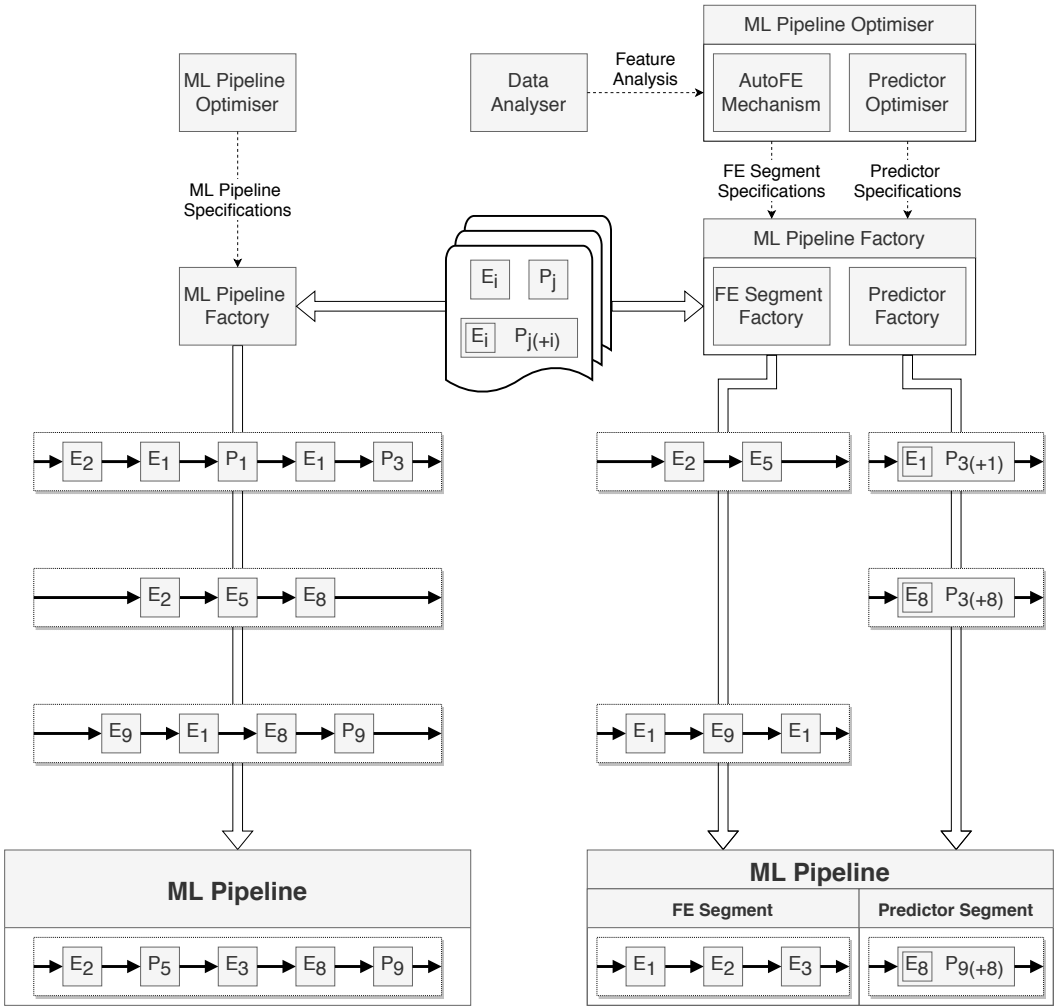


Fig. 8. A demonstration of two different approaches to ML-pipeline search, with E_i and P_j representing pools of feature-engineering (FE) and predictor ML components, respectively. ML components marked as $P_{j(+i)}$ are complex predictors that internalise FE transformation E_i . The left-side approach composes candidate pipelines freely, preferring simple predictors. The right-side approach subsections pipelines between FE components and complex predictors, selecting segments independently. The right-side ML-pipeline optimiser is provided the feature analysis of data to support ‘filter-type’ AutoFE mechanisms. Dashed arrows depict control and feedback signals. Solid arrows depict dataflow channels. Block arrows depict the transfer of ML components/pipelines.

This immediately opens up FE-heavy ML pipelines to optimisation approaches discussed earlier. However, despite the overlap, there are still important nuances to consider. Broadly speaking, MCPS studies are motivated by extending HPO to extreme configuration spaces, without fixation on what those components are. For instance, defining a predictor to be a mapping onto response space \mathcal{R} , and thus being evaluable with respect to expected outputs, an ideal MCPS optimiser would have no problem searching arbitrary ML pipelines with early-stage predictors, as exemplified by the

left side of Fig. 8. These candidates are not all invalid or suboptimal either; chaining predictors into stacked ensembles, as suggested by Fig. 7, may ultimately be the best solution for an ML task. Admittedly, in practice, a large fraction of MCPS research does deal with structural constraints, but these are a secondary consideration in service of making optimisation feasible.

In contrast, AutoFE is grounded in structural constraints, accepting the traditional notion that FE and prediction are unique and well-ordered sections of the data-processing sequence. In light of how intractable unconstrained ML-pipeline search can be, this attitude is not unreasonable. It is convenient to clump together both algebraic transformations and an expansive search through them under one feature-generation banner, just as it is convenient to have one monolithic process for combinatorial feature-subset selection. In light of this, the AutoFE approach can be described abstractly as an attempt to subdelegate the responsibility for MCPS optimisation, as demonstrated by the right side of Fig. 8.

However, because ML-pipeline optimisation is so novel, the concept of partitioning is under-explored. For example, a CNN is considered a complex predictor, where the initial layers are typically responsible for identifying features within images. The question arises: should these layers remain subject to a NAS procedure, or should they be selected/optimised externally as part of an AutoFE process? Expressed in another way, given three consecutive indivisible operators, i.e. $U : \mathcal{Q} \rightarrow \mathcal{F}_1$, $V : \mathcal{F}_1 \rightarrow \mathcal{F}_2$ and $W : \mathcal{F}_2 \rightarrow \mathcal{R}$, should V be targeted by FE optimiser Ω_{FE} or predictor optimiser Ω_{Pred} ? Ideally, there would be no difference, assuming that both optimisers treat V as a degree of freedom, have access to the same CASH-related search space, and have perfect information regarding the evaluation metrics of the $W \circ V \circ U$ composition as a whole. In actuality, these assumptions are rarely true, leading to the following inequality:

$$\Omega_{\text{Pred}}(W \circ V) \circ \Omega_{\text{FE}}(U) \neq \Omega_{\text{Pred}}(W) \circ \Omega_{\text{FE}}(V \circ U). \quad (3)$$

In essence, it is not simple to decide whether the onus of wrangling a topological transformation, $\mathcal{F}_i \rightarrow \mathcal{F}_j$, should fall to AutoFE or predictor design.

Regardless, despite the ambiguities around ML-pipeline segmentation and the generality invited by the MCPS paradigm, hard boundaries between pre-processors and predictors remain heavily ingrained within the ML community. Many of the MCPS-based AutoML systems discussed in Section 4 either implicitly or explicitly subdivide ML-pipeline search, as conceptualised in Fig. 8. As a result, jointly optimising both segments is considered somewhat novel [343]. Additionally, with priority focus on selecting classifiers/regressors, many systems also lean towards complex predictors, where feature transformations are treated as an embedded hyperparameter rather than a unique and free-floating component. The right side of Fig. 8 implies this; $P_{j(+i)}$ is a monolithic ML component, even though it represents P_j learning from a data remapping performed by E_i . This internalisation of feature-space transformation is not an intrinsic weakness, with the SVM kernel trick providing a well-known example of where it proves useful.

Notably, although subdividing ML-pipeline search space trades off generality for manageability, this may not be as limiting to AutoFE as cell-based search may be to NAS. Pre-processing does tend to have a natural order, with data-cleaning operations typically required at the start of an ML pipeline. While there are several ways to, for instance, choose how to deal with missing values, with the options possibly represented by a tunable hyperparameter, there is no point testing ML-pipeline compositions with a late-stage imputation operator. Their validity, or lack thereof, can be inferred immediately. Accordingly, AutoFE very much profits from further structural constraints, perhaps in the form of the HTNs that were introduced in Section 4. In anticipation of ensembled ML pipelines, discussed in Section 8, it is also worth highlighting that the pre-processing operations chosen for one composition are unlikely to differ from those for another, at least in relation to cleaning procedures. Any robust implementation of AutoML should consider working with pointers and

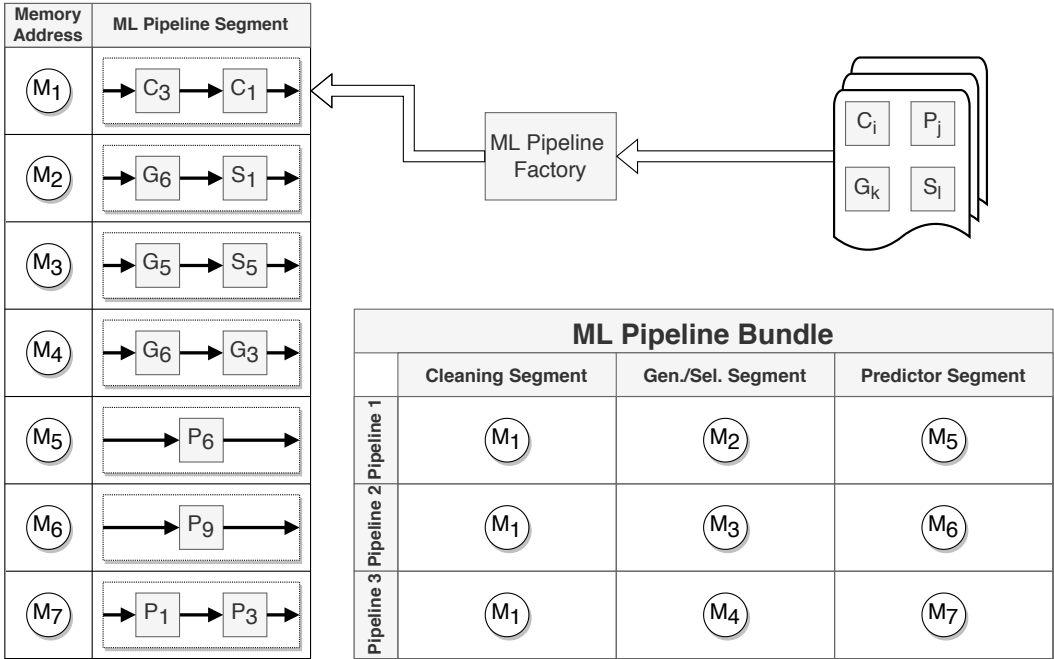


Fig. 9. An example of sharing ML-component compositions across ML pipelines via pointer/reference. Label C_i represents data-cleaning operations, P_j represents predictors, G_k represents feature generation, and S_l represents feature selection. Solid arrows depict dataflow channels. Block arrows depict the transfer of ML components/pipelines.

references, so that, as Fig. 9 demonstrates, the optimisation of one pre-processing segment carries over to all ML pipelines.

Setting aside the discussion of how it fits within a larger AutoML environment, the automation of FE has been a research interest for several decades. The concept of deriving new data attributes from an existing set has taken several names; for instance, alongside ‘feature extraction’ [286], there was also ‘constructive induction’ [253]. Much of the research associated with this topic, especially in the late 1980s and 1990s, is couched in the language of logic and expert systems, with feature transformations often applied in the form of Boolean operators [273]. Even so, several frameworks were developed early on in this space with the intention of automating feature generation via the chained application of constructive operators. The CITRE system is one such example, learning a decision tree from one collection of features to suggest a novel alternative set of logically compounded attributes, before retraining decision trees from the new feature set and recycling this process until converging to an optimum [248]. Although some of its contemporary frameworks would either ask a user to manage or just outright ignore the ballooning of the constructed feature set, the CITRE system was also among a few that pursued autonomous operations by ranking/pruning, with judgements of ‘quality’ based on information-theoretic measures, e.g. how usefully discriminative a new attribute is. This form of AutoFE proved beneficial in both accuracy gains and structural simplifications for the decision-tree predictors that were worked with.

Naturally, while some efforts focussed on how best to construct new features, optimal subset selection received its own attention. Central to this is the problem of determining which attributes of a data instance are most relevant to classification/regression. Excessive numbers of features

can significantly slow down and worsen ML-model training via the curse of dimensionality, and correlations/redundancies can outright destroy the performance of certain predictors, e.g. Naive Bayes classifiers. By the late 1990s, feature-selection methods were roughly categorised under ‘filter’ and ‘wrapper’ approaches [203]. Filters involve determining useful attributes via dataset analysis, with no regard for the subsequent predictor. The aforementioned CITRE system somewhat exemplifies this, even though its quality-based filtering is technically embedded into the training of a decision tree. Not unexpectedly, surveys of the time list and debate many ways to assess the relevance of a feature, e.g. the degree of unique correlation between class labels and the values of a feature [41]. In contrast, wrappers evaluate whether a feature is useful based on the error rate of a subsequent predictor. Thus, feature-selection filters can be considered fast and generic but also hit-and-miss, while wrappers are slow but accurate. As an aside, the right side of Fig. 8 acknowledges the filter strategy by allowing advisory feature analysis to be received by the ML-pipeline optimiser.

In any case, while hardware and procedural details have improved over the decades, the overarching concepts of automating FE generation/selection have generally remained the same. Instead, pools of feature-constructing components have been expanded over time, exemplified by the Feature Incremental ConstrUction System (FICUS) [246], which applies selection in a filter-based manner. Similarly, the Feature Discovery algorithm (FEADIS) promotes the inclusion of periodic functions, also serving as an alternative example of a greedy wrapper-style system, i.e. one that checks whether each newly proposed feature improves predictor performance [85].

More recently, as of 2015, the Data Science Machine (DSM) and its Deep Feature Synthesis algorithm have acquired somewhat of a pioneering status in the modern AutoML wave, possibly due to their performance in several ML competitions [192]. The DSM is presented as an end-to-end system, thus going beyond AutoFE and also applying Bayesian HPO to a random-forest predictor. As an AutoML system designed for extracting features from relational databases, it served as an inspiration for the ‘one-button machine’ (OneBM), which focussed on extending relational graphs to unstructured data [217]. This thread of research later led to training relational RNNs rather than searching pools of pre-specified operators for the purpose of representing optimal feature-generating transformations [216]. The DSM system was also acknowledged in the release of ExploreKit, which works with non-relational tabular data instead and, reminiscent of wrapper-based FEADIS, uses a pool of general operators to build up its features [194]. It would, in turn, serve as a direct inspiration for the ‘Feature Extraction and Selection for Predictive Analytics’ (FESPA) method, which claims distinctiveness by implementing regression rather than classification, while also aiming to keep feature generation and selection as well-separated processes [345].

As previously implied, AutoFE is dogged by the computational complexity of searching such a fine-grained space of operators, especially as most modern systems are wrapper-based and require significant time to train their predictors per pipeline evaluation. Neural networks have certain go-to methods, with auto-encoders sometimes employed to find compact feature-space representations of data instances, e.g. in a competition studying online course dropouts [71]. However, in both deep-learning and general contexts, research attempts explore optimisation methods to hone more efficiently onto optimal FE pipelines. Evolutionary algorithms are one such approach for feature selection [372], with genetic programming being an obvious choice for chains of operators, in similar fashion to MCPS approaches discussed in Section 4. Example implementations have been proposed for AutoFE in specific applications, like compiler-based loop unrolling [218], or in general contexts, like with DNNs [155]. Unsurprisingly, reinforcement-based strategies such as Q-learning have also been proposed, noting that the DSM and contemporaries are often bogged down by searching unnecessary portions of feature-transformation space [197]. Beyond optimisation, other mechanisms have also been explored to boost efficiency. For instance, a tool called Zombie

groups raw data by similarity prior to processing, so that useful data instances are prioritised by feature-generation code and, presumably, subsequent incremental learners [14].

To date, AutoFE approaches have been applied in various domains. Natural language processing (NLP) is often more challenging than tabular data to generate features for, but efforts have been made to automatically extract features via entity-entity relationships present in semantic knowledge bases [66]. Similarly dealing with NLP, RaccoonDB is a system that accelerates feature extraction from social media for nowcasting [14]. These approaches often deal with immense online sources of data, which are not practical to load into local memory, and it is an open question how best a general AutoML implementation should interface with these inputs. As for tabular formats, wrapper-based feature extraction has been explored in the context of condition-based aircraft maintenance, where GA-based search seemed to be the best performing out of several tested optimisation routines [134]. In the clinical setting, as part of designing a ‘prediction tool using machine learning’ (PredicT-ML), features have been condensed from datasets via numerous temporal aggregations and operators, subsequently selected in a filter-based manner with metrics such as information gain [239]. Elsewhere, a defensive publication has proposed using decision-tree techniques for feature selection in the context of training neural networks to predict data-centre outages [70]. This diversity of applications suggests that perhaps AutoFE may be more valuable to the broader community than predictor-specific CASH.

To conclude this section, it is worth mentioning that many modern AutoFE approaches do not search for an optimal FE pipeline segment from scratch. For instance, the Cognito system, a framework using greedy exploration to traverse a tree of feature transformations [198], all prior to adopting an RL-based strategy [197], eventually implemented a ‘learner-predictor’; this module uses historical datasets to recommend FE transformations [196]. Similarly, ExploreKit [194] and FESPA [345] both augment their AutoFE processes with prior knowledge, if available. Moreover, there are recommender systems based on historic datasets that solely target feature generation [260] or feature selection [272, 356]. This idea of leveraging experience from beyond a current ML task is an attractive one, its potential benefits not limited to just AutoFE.

7 META-KNOWLEDGE

Thus far, the previous sections have framed AutoML and the search for a task-optimal ML model as an optimisation problem. Whether in the context of an ML component or the context of an ML pipeline, whether for NAS or for AutoFE, many fundamental advances of this past decade have revolved around how to represent complex search spaces, how to constrain them appropriately, and how to apply search strategies efficiently. This focus is not surprising; optimisation is arguably the purest mechanism for identifying a good ML solution. It was certainly acknowledged as one way to approach algorithm selection, back when this problem was posed in the 1970s [286]. However, even in the modern era, training/testing each iteration of proposed ML model can take substantial time. So, given the theory/hardware limitations of the intermediate decades, making large-scale optimisation infeasible beyond ranking small sets of candidate solutions, ML practitioners had to rely on different tactics. In particular, the concept of meta-learning was popularised, mechanically leveraging knowledge gained from previous learning processes to support ML-model selection for a new task [44]. As a form of AutoML before the ‘AutoML’ abbreviation was coined, and briefly supplanted by communal interests in optimisation, meta-learning has reacquired status as a potentially empowering upgrade to standard model-searching systems [7, 45].

The core foundations of meta-learning were motivated long before the term itself entered common parlance. For instance, when algorithm selection was initially codified as a research question, one proposed angle of attack was to find ways of classifying and categorising ML problems, thus identifying similarity-based groupings for which ML models/algorithms would be particularly

effective [286]. This appeared to be a solid strategy; any hope in finding one super-algorithm that would be maximally performant for all settings was dashed by the publication of the no-free-lunch theorems in ML [368] and optimisation [367] during the 1990s. In essence, the theorems state that the average performance of all algorithms across all problem domains are equivalent. Algorithm selection can only be biased towards high performance by linking both a new ML problem and its context to prior experience and then acting on this ‘meta-knowledge’. This form of ‘learning to learn’ appears biologically justified, with studies of pre-schoolers identifying that humans acquire the capacity to “search for underlying commonalities” at an early age [49].

However, as with previous sections, first a caveat: terminology is in flux and the boundaries of the meta-learning topic evolve over time. To add to the confusion, ‘meta-learning’ has sometimes been used to describe ensemble-based approaches, where the ‘meta’ prefix refers to classifiers composed of classifiers [58]. Even disregarding this obvious semantic discrepancy, multiple communities have taken the algorithm-selection problem and related meta-learning approaches in different directions, developing inconsistent terminology and arguably suffering from a lack of interdisciplinary communication [316]. Fine tweaks to definitions are also not uncommon; whereas meta-learning has always involved the transfer of meta-knowledge from different domains/problems, learning from a previous training run on the same dataset is now also considered under the same umbrella term [45, 223].

As a process, while meta-learning is often employed in one-off research investigations, there have been several proposals for how to encase its automation within an idealised general architecture [142, 175, 177, 353]. In similar vein to the illustrative framework developed in this paper, these architectures aimed to systematise the principles underlying a preceding flurry of published systems, where the implementations were based on the Knowledge Discovery in Databases (KDD) process [104], a template for data exploration. Indeed, while KDD is a broad topic and associated toolkits are varied, many intelligent discovery assistants (IDAs) were designed to support the construction of data-processing pipelines via meta-learning principles, often providing recommendations by analysing datasets and relating them to previous experience. They have been surveyed and reviewed extensively elsewhere [2, 313, 348]. Importantly, several were even capable of automatically recommending predictors, despite the lack of optimisation, thus acting as AutoML forerunners. These days, pure IDAs appear to have waned in relative importance, with the role of meta-learning evolving to, primarily, support CASH [349]. This HPO-driven paradigm shift in the way AutoML uses meta-learning is made stark when comparing relevant workshops for the 20th and 21st European Conferences on Artificial Intelligence (ECAI), held in 2012 [350] and 2014 [351], respectively.

Before discussing meta-learning in depth, we provide an illustration, via Fig. 10, of how the concept may be integrated into an AutoML framework. In effect, this schematic is another sequential upgrade of the CASH-focused system depicted in Fig. 5, expanding beyond the inclusion of ML pipelines that was shown in Fig. 6. The data-analysis module and its associated feature analysis, as recommended by Fig. 8 to support subdelegated AutoFE optimisation, are also present. However, for this upgrade, the proposed analysis modules for ML tasks and data sources have the additional responsibility to extract metrics for task/dataset similarity. A library of ‘meta-models’ stores the accumulation of previous experience, and, according to how recognisable the current context is, can provide an AutoML system with advice. Typically, this comes in one of two forms: solutions that worked well previously, or ways to find solutions that worked well previously. Logically, these suggestions are best sent to the ML-pipeline planner or the ML-pipeline optimiser, respectively. Additionally, an ideal AutoML system should not just leverage meta-knowledge but also contribute to it. Hence, the optimiser, which has a view of its own performance along with that of any candidate solution, is charged with passing back results for the sake of development.

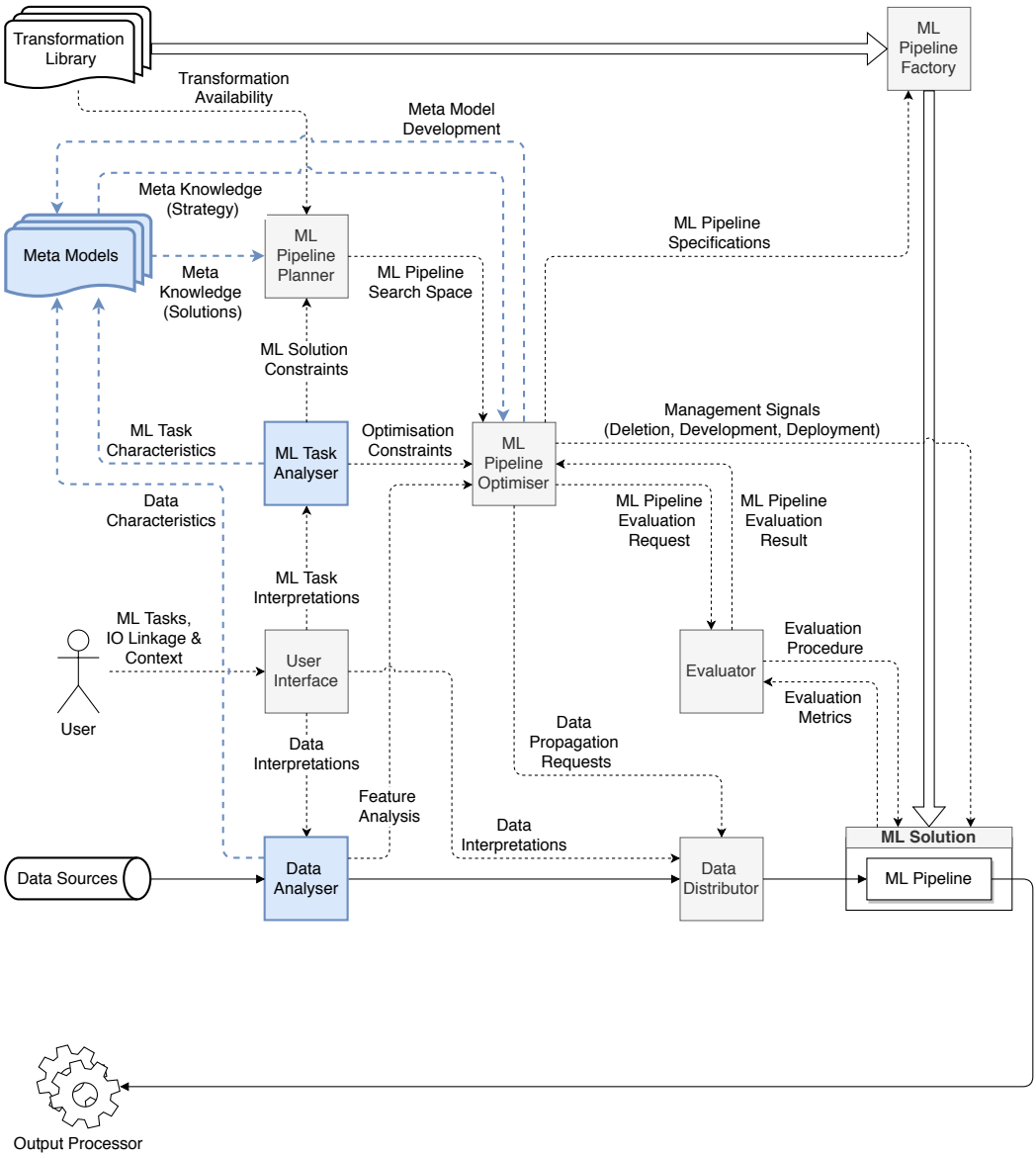


Fig. 10. (Colour online) A high-level schematic of an improved AutoML system that employs meta-learning for search-strategy and solution recommendation. This architecture is an upgrade to that of Fig. 5, with blue highlights emphasising the main new additions related to meta-knowledge. Dashed arrows depict control and feedback signals. Solid arrows depict dataflow channels. Block arrows depict the transfer of ML components/pipelines.

Naturally, the usual disclaimer applies, in that Fig. 10 does not necessarily represent the only way to design a functional AutoML system. Critically, though, while CASH is unlikely to vary dramatically in integration, meta-learning represents a methodology as opposed to a method and is thus much harder to conclusively encapsulate. Any module, from UI to data distributor, could

conceivably be designed to improve based on previous experience. Nonetheless, given that the vast majority of published research examines meta-knowledge in the context of ML-solution search, Fig. 10 is sufficient to abstractly represent most approaches. Discussion around related limitations is thus reserved for Section 14.

When it comes to the details of meta-learning, the most common tactic is to seek similarity between datasets. Characterising them by useful ‘meta-features’ is thus a necessary challenge and has been extensively explored. Obvious metrics include directly observable descriptors, e.g. the number of data instances or features, as well as statistical measures, e.g. variances, and information-theoretic meta-features, e.g. attribute entropy [130]. For supervised-learning classification specifically, complexity measures aim to categorise a dataset by the nature of its separating boundaries, quantifying feature overlap, class separability, and geometries/topologies/densities of class-spanning manifolds [158]. Analogous complexity measures have been proposed in the context of regression problems [237]. Naturally, data contexts that contain more structure provide more categorically supportive information, with discriminative characteristics proposed for both time series [114, 225, 226] and network-oriented datasets [291]. These latter meta-features were later employed in an AutoML system specifically focussed on biological ecosystem networks [27]. However, it should be noted that most meta-features described thus far arise from operations on the data itself. One ML-model recommender named AutoDi explores a different approach [344]; alongside ML algorithms, datasets are assigned textual descriptors in the form of embedding-vectors that are derived from online corpora, e.g. academic papers and Wikipedia.

In similar fashion to the filter/wrapper dichotomy in AutoFE, some strands of research have considered whether a prediction process should be involved in characterising an ML problem. For instance, several efforts have explored training archetypal predictors such as decision trees on a dataset, so as to use the model-based properties for dataset characterisation [30]. More recently, in the context of automating semi-supervised learning, where not all data instances are labelled, outputs of clustering algorithms serve the same purpose as the aforementioned decision trees; cluster-based statistics of those resulting models are used as ML-task meta-features [232].

More common than describing a dataset by the properties of a trained ML model is describing it by the performance of that model. This approach, given the name ‘landmarking’ in 2000 [276], can be risky if an ML algorithm is overly stochastic in the model it produces. However, it is argued that simple high-bias learners like linear discriminants or decision stumps are quick and robust estimates of ML-problem complexity [115]. These landmarks have thus found use alongside traditional dataset meta-features within meta-learning systems [54]. It is worth mentioning though that a ‘sampling-based landmark’, seemingly related, is an alternative that involves training a non-simplified ML model on a subset of data [319]. In effect, partial learning curves of accuracy versus sample size for sampling-based landmarks can be used to further characterise datasets, but these curves also contain additional information about the relative performance of candidate ML algorithms, allowing performance estimates for a full dataset [220, 221]. Moreover, knowing how ML algorithms compare on one dataset can be used to guide ranking strategies on another similar dataset [222]. This idea has been exploited several times, with one study working to meta-learn pairwise comparisons of utility between ML algorithms [138]. It is also related to collaborative-filtering approaches, where patterns of model performance are assumed to correlate between similar datasets [315, 328].

Once a set of meta-features have been selected for a class of ML problems, with each problem generally corresponding to an appropriate inflow dataset, a standard practice for meta-learning is to construct a meta-model, i.e. a function from dataset meta-features to a recommendation variable, usually trained by ML. The development and use of such meta-models are depicted in Fig. 10. Often, a k-nearest neighbour (kNN) approach is used to leverage similarity in simple fashion, sometimes

called an instance-based meta-learner [73], while deriving decision rules is another simple option for sparse metadata [4]. However, the meta-model can be as sophisticated as desired. An archetypal example of meta-model construction is depicted within a study that evaluated optimal SVM kernels for 112 classification problems and subsequently trained a decision tree as a meta-classifier to map vectors of dataset characteristics to those optimal kernels [10]. Just like a standard ML model, the meta-model could then be passed a query, i.e. a new dataset transformed into a set of meta-feature values, and would then calculate a response, i.e. a suggested SVM kernel. In similarly model-specific contexts, recommenders have been trained for other SVM hyperparameters [139] and for decision tree induction [141].

In the modern HPO-centred context of ML model/algorithm selection, meta-learning is commonly used to suggest a good initial point to start searching from. This process is called warm-starting, whereby a systematic start-from-scratch optimisation procedure is boosted by an externally derived heuristic. It is represented in Fig. 10 by the solution-recommending meta-knowledge signal being propagated on to the optimiser. This concept of warm-starting is exemplified by a study wherein GA-based HPO is applied to a meta-learned initial population of hyperparameter values [284]. It has also occasionally been incorporated within fully implemented AutoML systems, e.g. Auto-sklearn [108]. However, there have been many nonstandard variations on the theme, such as recommending a warm-start candidate by first minimising across a weighted combination of previously encountered HPO loss functions [363].

Meta-learning has been leveraged in several other unique ways to support HPO. For instance, the Automated Data Scientist distinguishes itself from Auto-sklearn by seeking to directly predict optimal hyperparameters, as opposed to suggesting a warm-starting configuration [265]. Elsewhere, meta-knowledge has been used to recommend regions of configuration space that should not be explored [362]. Then there are proposed upgrades to SMBOs in particular, where the underlying surrogate functions are constructed across all encountered datasets rather than per each new ML task [26]. The original proposal was later improved upon to deal with a scaling challenge, i.e. the fact that evaluation metrics are not directly comparable across ML problems [377]. Variants of these concepts have been developed in an adaptive manner, exploiting external knowledge for SMBO while a new solution space is unknown, before gradually shifting to an acquisition function based on local knowledge [364]. Whether the meta-learned surrogate should be shared or ensembled across all ML tasks has also been debated [366]. Again, all of these variant approaches underscore how difficult it is for a proposed AutoML framework to encapsulate the full range of meta-learning approaches, considering that prior knowledge can feed informatively into operations at almost any level.

Because meta-models have no particular constraint on their recommendation variable, provided that every ML problem in meta- \mathcal{Q} -space maps to an independent response value in meta- \mathcal{R} -space, meta-learning has been used to suggest elements of ML solutions beyond standard sets of hyperparameters. For instance, recommender systems have been designed for classifiers, both standard [46] and multi-label [63], as well as regression-based predictors [266]. Admittedly, this is not a particularly radical extension in light of the CASH paradigm, which treats ML algorithms as hyperparameters. Regardless, as Section 6 hinted, FE operators are also among meta-learned ML components [194, 196, 345], whether generative [260] or selective [272, 356]. Certainly, a recent empirical study espouses the use of a meta-model for pre-processor selection [307], and the PRESISTANT system is an example of a modern IDA centred around this aspect of AutoML [38, 39]. Moreover, recommender systems can even propose entire ML pipelines; Meta-Miner is one such system employing a meta-model that maps dataset meta-features to KDD-workflow characteristics, the latter serving to encode information relating to pipeline structure [262].

Further emphasising how meta-knowledge can be employed at any level of an AutoML architecture, entire ensembles have been recommended based on standard and landmarker meta-features [207], with the topic of ensembling elaborated in Section 8. Likewise, in the context of adaptation, further described in Section 9, meta-learning has proved useful in selecting the active predictor within a heterogenous ensemble [173]. Even the parameters of an optimiser have been subject to tuning processes; notably, for CASH-solvers, these sit at a higher level than parameters/hyperparameters for ML models/algorithms. While there are direct strategies for automating optimiser-engineering, such as using RL to learn an update policy while the optimisation is running [229], there are also approaches based on standard meta-models [3] or trainable neural architectures, e.g. Long Short-Term Memory (LSTM) networks [15]. These effectively transfer settings for quickly convergent optimisers across similar problems. Naturally, searching parameters for parameter-searchers can become a recursive problem, and it is an open question as to how much of an AutoML system should be automatically tuned.

As a side note, while most meta-models work towards ML model/algorithm selection on the basis of predictive accuracy, there are other metrics for evaluating ML pipelines; see Section 10. In particular, the issue of ML-algorithm runtime has been visited several times within the field of meta-learning. In one direction, runtimes have been folded into meta-features, so that learning curves [319] become loss-time curves [347]. A ranking scheme for ML algorithms based on these ideas proved competitive with Auto-WEKA, especially given small time budgets for which the AutoML system could not fully exploit its optimisation capabilities [53]. Alternatively, in the opposite direction, meta-knowledge has been deployed to predict runtime [172]. One series of research efforts has revolved around meta-learning the complexity of Bayesian network-structure learning (BNSL) problems, which relates to BNSL-solver runtime [245], and has also contributed to the 2017 Open Algorithm Selection Challenge; the task here was to meta-learn a scheduler to solve an unseen problem, given the time required to observe features and run solvers on previous instances [244].

At this point, it is worth discussing transfer learning, given that boundaries between the topic and meta-learning can be inconsistent and ill-defined in the literature, if not outright nonexistent [271]. Generally, transfer learning does not learn a meta-model mapping of an ML problem to a recommendation variable; it simply copies the value of a recommendation variable, i.e. knowledge, from one ML problem to another, e.g. an entire cell of CNN layers. There is an implicit assumption that the source/target ML problems are similar, but there is no meta-model around to quantify this, hence why a transferred variable, e.g. the CNN cell, must usually be tuned afterwards, while an ideal meta-model could theoretically recommend a value that immediately accounts for differences in setting. Some ML practitioners may discuss semantics further, perhaps constraining the scope of the knowledge that can be meta-learned or transferred, but we avoid this debate. From the perspective of architectural design, transferrable knowledge is already contained within the meta-model library in Fig. 10, offered to the ML-pipeline planner as solution advice in the form of meta-knowledge, provided that an ML task at hand shares an identical and usually high-level characteristic to one encountered previously. For example, if two tasks focus on text translation, it is circumstantially reasonable to copy and paste a neural model from one to the other, essentially as a warm start, even if the human language involved varies.

Transfer learning is commonly associated with deep learning in the current day and age, often referring to the practice of using a neural network that is pre-trained for one setting as part of a new DNN, thus transferring useful feature representations. One study aims to support this domain adaptation by meta-learning the relation between dataset dissimilarity, based on meta-features, and the amount of tuning required for a transferred DNN substructure [8]. Transfer learning has also been proposed to extend RNN-based NAS [386], although the concept is somewhat different and

involves augmenting the RNN controller with a task-embedding strategy; the similarity of ML tasks when converted into embedded vectors drives correlated child-network design sequences [369]. For a general AutoML system, it is also of interest whether feature representations, i.e. FE pipeline segments, can be transferred from unlabelled and even arbitrary data to supervised learning tasks. This idea has been previously explored under the name of ‘self-taught learning’ [281].

Given the current popularity of DNNs, many novel meta-learning advances are often framed within such contexts. Model-agnostic meta-learning (MAML) was proposed in 2017, where a network is trained across a set of similar ML tasks so that, when used to initialise a gradient-based search for a new ML task, few iterations are required to optimise the model [109]. In effect, good initial DNN weights are meta-learned for rapid few-shot learning. The utility of MAML has been demonstrated in several contexts, e.g. efficiently learning an RL policy for robotic control [110].

Ultimately, meta-learning provides sound foundations for upgrading almost any element of a modern AutoML system, at least in theory. The motivation is that, assuming previous experience is relevant to an ML task at hand, it would seem beneficial to incorporate meta-knowledge in data-based predictive/exploratory systems. Unsurprisingly, there is evidential support for this view; one strand of research involving a ‘meta-mining’ module [262] and an IDA named eProPlan/eIDA [200] found that the ‘best’ workflows suggested by KDD system RapidMiner, specifically ML pipelines composed of frequently used operators, were significantly outperformed by workflows ranked and recommended via meta-models. In effect, automation based on meta-learning generally produces better ML solutions than those manually selected by human practitioners. Nonetheless, the extent of its effectiveness cannot be assumed. Leveraging meta-learning requires several unclear design choices, so much so that meta-model selection has itself been the focus of meta-learned recommendation [73]. Concern has also been raised that standard sets of meta-features cannot seem to discriminatively subdivide the space of ML tasks according to pre-processing pipelines they should employ [131]. Then there is the issue that training a recommender system well requires significant amounts of data and experiments from similar contexts, which may simply not be available in practice. It is not even clear whether the obvious solution to this, i.e. drawing meta-knowledge from other domains, is particularly effective [9]. In essence, the promise of meta-learning is there, but, as with many AutoML research threads, more critical analysis is required to properly evaluate the benefit of employing these mechanisms.

8 ENSEMBLES AND BUNDLED PIPELINES

Every predictive ML task, i.e. where the response space \mathcal{R} is previously defined by a user, must involve a predictor at some stage, namely a transformation that converts some feature representation of queries in \mathcal{Q} over to responses in \mathcal{R} . It can be a weakness, however, for an ML model to rely on only one predictor. An AutoML system may not have access to a strong enough learner in its pool of ML components, or, alternatively, perhaps each predictor is too prone to overfitting for a particular dataset. Worse yet, for continuous streams of data, relying on one predictor is especially brittle. True AutoML needs to be efficiently adaptive, easily maintained even when one ML model fails. Thus, for many reasons, managing a multiplicity of ML pipelines and their ensembles is an important thread of research.

Aggregating multiple models is a powerful technique for controlling error. One review into forecast combination provides a Laplace quote to suggest that this was understood in the early 1800s [69]. Thus, in the modern era, a lot of discussion about ensembles is framed within the context of bias-variance decomposition (BVD), which was introduced to the ML field by the 1990s [129]. In essence, the theory states that the failure of an ML model to mimic a desired function arises from three error terms:

- Bias – The inability of the ML model to fit data due to its simplifying assumptions. High bias is typically the cause of underfitting and commonly describes weak learners.
- Variance – The inability of the ML model to generalise, given how much its fit changes for different data samples. High variance is typically the cause of overfitting.
- Noise (or Bayes error) – An irreducible error related to how poorly sampled data represents the desired function.

Typically, BVD theory is often brought up in the context of ML-model limitations and bias-variance trade-off, although there is ongoing debate about where exactly this trade-off is applicable [261].

In terms of ensemble methods, there are three common types that find frequent use:

- Boosting – An ensemble is constructed in sequential manner, with each new predictor training on re-weighted data; these weights prioritise data instances that were poorly modelled by previous predictors. The output of the ensemble is usually provided as a weighted average across predictors. This entire process often reduces bias, i.e. underfitting.
- Bagging – An ensemble is constructed in parallel manner, with each predictor training on data sampled with replacement. The output of the ensemble is usually provided as an average across predictors. This entire process, also known as bootstrap aggregating, often reduces variance, i.e. overfitting.
- Stacking – An ensemble is constructed in layered fashion, where the outputs of base predictors are appended to their inputs. The next layer of predictors trains upon the concatenation produced by the previous layer.

Many ensemble methods are often homogeneous, i.e. based on one predictor, with boosting/bagging often incorporating weak learners. However, heterogeneous ensembles have been proven effective in their own right [71], perhaps even more so [127]. Ultimately, the predictive power of an ensemble is bound very closely to the diversity of its constituent learners. Ensuring independence between predictors, or even complementarity via negative-correlation learning [91], can have strong performance impacts.

Notably, ensembling is not a foreign concept to AutoML development, with many systems being promoted specifically because of these strategies. For example, employing mlrMBO [40] as a CASH-solver, one effort automates gradient boosting [336]. Given that many AutoML packages search through a pool of ML models/algorithms, the resulting ‘autoxgboost’ implementation is relatively unique in focussing on a single learner. Elsewhere, stacking methodologies have seen considerable uptake, utilised in the BO-based Automatic Frankensteining framework [365] and the GA-driven Autostacker [64]. Most recently, AutoGluon-Tabular joined this list, suggesting somewhat counter-intuitively that ensembling techniques could be more important than solving CASH for achieving state-of-the-art accuracies [98]. Indeed, a common theme within related publications is that ensemble-based systems frequently appear to outperform their non-ensembled peers, with the AutoGluon release even providing an ablation study to emphasise the benefits of multi-layer stacking. However, the ablation study does not seem to account for any possible advantages afforded by manually well-selected hyperparameters, which could provide other AutoML systems a commensurate runtime discount. Further analysis with good benchmark design will be required to settle the debate.

What is clear is that the automation of ensembled ML models cannot succeed without well-developed strategies for selection and combination. Coalescing results across multiple ML predictors generally appears more beneficial than arbitrating amongst them [58], although this assumes that each predictor is still informative in some way regarding the inflow-data distribution. This is not necessarily true for concept drift; see Section 9. It also remains an open question regarding what the most effective way to build an ensemble is, with one study suggesting that greedily attaching

heterogeneous ML models on the basis of individual performances can outperform boosting and bagging [56]. Unsurprisingly, ensemble selection can also be treated as a variant of the CASH problem. Certain SMBOs have been updated to automatically produce ensembles [215, 228], with associated extensions into regression problems and upgrades for dynamic ensemble-sizing [292]. Likewise, GA-based methods have also been explored for ensembles [120]. Genetic programming has been applied to automatically evolve forecast-combination structures for airline data [227] and has been further investigated alongside meta-learning strategies [207]. In fact, there exists an entire genetic-programming framework named GRAMmar-DrIven ENsemble SysTem (GRADIENT), which automatically creates combination trees of base-level predictors [340]. A two-tier approach would later be proposed as an alternative to this, where high-level ensembles act as fuzzy combinations of low-level ensembles [341].

Two-tier ensembling, alongside its multi-layer generalisation, is worth further discussion. While a lot of research has traditionally been invested into flat weighted ensembles, certain studies, both theoretical [294, 295] and empirical [296], have identified that the performance limits of a multi-classifier model can be stretched by suitably structuring the model into ensembles of ensembles. The other upside of working with such ‘deeper’ ensembles is the flexibility of delegation that they possess, promoting ML solutions that leverage specialisation. The idea is that each learner within a collective, potentially an ensemble of lower-level learners in its own right, can be trained to specialise on its own non-overlapping selection of data, e.g. by instance or feature [120], according to the requests that an ensemble manager makes of a data distributor. Multi-level ensembling thus allows for a much more controlled and targeted synthesis of information. Of course, specialisation must still be managed carefully, with one word-sense disambiguation study showing that ensemble accuracy is very sensitive to the optimisation of individual experts [162]. Moreover, designing a multi-layer MCPS is not trivial, relying on the right approaches for local learning and ensemble diversity [5]. However, if done right, specialisation can be very powerful. This is evidenced by a series of publications exploring multi-layer ensemble models applied to airline revenue management [287–290], which, due to seasonal variations in demand, also investigated the dynamic evolution of these structures. Adaptation is further discussed in Section 9. Similarly themed considerations likewise resulted in a generic multi-level predictor system for time series that managed to achieve significant performance in several contemporary forecasting competitions [297].

In terms of the illustrative AutoML framework developed in this work, many of the elements discussed thus far already cater to ensembled approaches. Homogeneous ensembles can be constructed/aggregated internally within an ML component with little additional effort, as hinted at in Section 2. Likewise, while heterogeneous methods cannot be represented on the ML-component level, a simple example of stacking has also been demonstrated, internal to an ML pipeline, within Fig. 7. However, we emphasise that the intent behind defining ML pipelines is to focus on enabling complex sequences of data transformations. There is still an implicit assumption that a one-pipeline ML solution must engage with the entire scope of a data source as part of the learning process, no matter the branching degree of its internal ML-component DAG. This is a severe limitation, as a diverse team of learners concentrating on different parts of a problem, i.e. specialisation, may be much more effective than a single learner with a scattered attention. Now, it is arguable that heterogeneous boosting, with its sequential nature of having learners clean up the mistakes of other learners, can still be squeezed into ML-pipeline representation with sufficiently elegant inflow-data control, i.e. constructing new candidate solutions as extensions of previous ones but training them on novel samplings. However, heterogeneous bagging relies on the distribution of dissimilar data samplings across various learners, all potentially working asynchronously.

The upshot is that an ML solution needs a higher level of structure beyond an ML pipeline to fully support heterogeneity and complex data re-distributions. This is best served by the ability

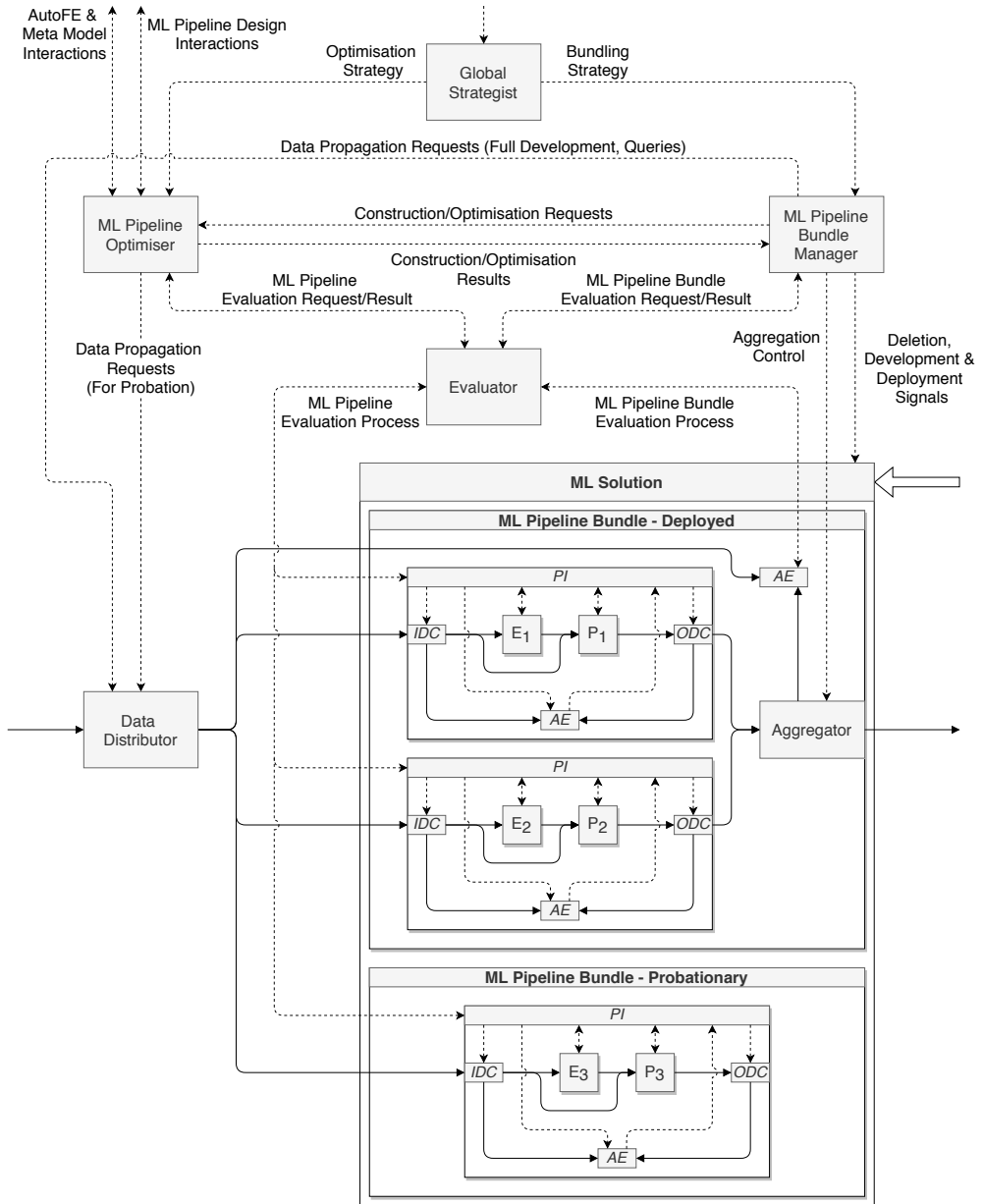


Fig. 11. A schematic of a deployed two-pipeline ML bundle acting as an ML solution within an AutoML system, with a third ML pipeline kept in its own bundle for model development. Only the deployed bundle requires aggregation for an actionable output, along with a final accuracy evaluator (AE). ML components marked E_i and P_j represent feature-engineers and predictors, respectively. Other abbreviations denote a pipeline interface (PI), inflow-data controller (IDC), and outflow-data controller (ODC). Dashed arrows depict data control and feedback signals. Solid arrows depict dataflow channels. Block arrows depict the transfer of ML components/pipelines.

to manage multiple ML pipelines simultaneously and even ensemble them together. There is no consistent name for this specific parallelisation; here, we call the arrangement an ML-pipeline bundle, demonstrating in Fig. 11 how the structure could be utilised within an AutoML system. Within this illustration, there is a new module that oversees ML bundles, deciding when to add, remove, develop or deploy ML pipelines within the arrangement. Only one bundle is ever deployed though, and this one needs both an aggregator for individual outputs and an associated accuracy evaluator. The bundle manager controls the aggregator, e.g. weighting the sum of outputs, and also has access to all evaluations that may assist its prescribed ensembling strategy. However, while the bundle manager does decide when a new ML pipeline must be added to the deployed ML bundle, possibly overwriting an underperformer, the pipeline optimiser is tasked with the actual construction and optimisation thereof, as expected.

Naturally, this division of responsibility does require additional modifications to the architecture. Data propagation requests are now split between the optimiser and bundle manager by whether an ML pipeline is under probation, i.e. in the process of being optimised/validated, or under deployment, respectively. In static ‘one-and-done’ ML processes, the entire ML solution will typically be wholly in a state of learning followed by wholly in a state of use, but the compartmentalisation depicted in Fig. 11 becomes important in dynamic ‘long-life’ ML; see Section 9. Furthermore, it is no longer sufficient to provide optimisation constraints upon interpreting/analysing a new ML task; strategies for both HPO and pipeline bundling must be determined. Within the illustrative schematic, these are prescribed by a global strategist. Further commentary on this is provided in Section 14.

Ultimately, choosing when to deploy an ensemble remains debatable. They can often improve the accuracy of a predictive ML solution markedly, but, as Section 10 discusses, the performance of an AutoML system cannot be judged on predictive error alone. Ensembles complicate ML models and become increasingly difficult to interpret, a challenge that regularly faces algorithm design and selection in the modern era [293]. One way to counter this is to train a simpler interpretable model to mimic the decisions of a more complex one, or use a more direct process of compression if available, and this concept of ‘knowledge distillation’ is a research topic in its own right. It has been heavily investigated, for example, in the context of fuzzy neural networks [92, 116, 117], with similar approaches tried for decision trees [93]. In relation to AutoML, distillation has also recently been explored with AutoGluon [100]. In the long run, the procedure may be worth implementing as one of several high-level processes to enact in the relative downtime of an autonomous system, i.e. seeking condensed versions of deployed ML pipelines. Regardless, the capacity to build ML-pipeline bundles/ensembles remains a recommended upgrade for any AutoML system that seeks to pursue long-term maintenance/adaptation schemes.

9 PERSISTENCE AND ADAPTATION

From here on in, all reviewed research is framed within the context of working towards *AutonoML*, not just *AutoML*. This distinction is drawn because, to date, the vast majority of *AutoML* research has focussed on automating the construction of ML models for fixed-term ML tasks. Broadly put, an *AutoML* system is engineered to be a powerful tool, turned off after use, whereas we contend that an *AutonoML* system should instead be viewed as an artificial brain, continuously developing while active. Unsurprisingly, many proposed upgrades for ML autonomy are biologically inspired [6]. However, conceptually upgrading *AutoML* to become self-governing is highly ambitious and certainly not a direct route. Thus, for practical purposes, we propose a fundamental characteristic to define an *AutonoML* system: the capacity to persist and adapt [383].

Persistence is the idea that an *AutonoML* system should be capable of multi-level operations in the long term. Technically, keeping a trained ML model under deployment is a trivial form of persistence that all *AutoML* systems are capable of. However, support for persistent learning

is much rarer. Few AutoML systems are designed for the streaming of inflow data, limiting ML operations to datasets of known finite size. This is understandable, as dealing with data from dynamic systems upstream of an ML process is very challenging, with the potential to incur a lot of technical debt [310]. On the other hand, the ‘Internet of Things’ (IOT) is a growing driver of big data analytics [80]; the need to embed ML systems into long-life dynamic environments is likely to increase in the coming years.

At a low level, streamed data is nothing new to ML, with varying types of online-learning algorithms in existence [43]. Moreover, many tunable data transformations that traditionally work in one-off fashion have incremental variants, such as PCA [147]. For instance, one SMBO proposal highlights its use of Mondrian forests as an online alternative to random forests [201]. There is even at least one entire software environment dedicated to supporting ML for streamed data, e.g. Massive Online Analysis (MOA) [37]. Thus, inevitably, researchers have begun studying the extension of AutoML processes into streaming contexts. This was the driving motivator for a 2018 NIPS competition into lifelong AutoML, as well as a recently released evaluatory framework named `automl-streams` [173].

Importantly, stream-ready AutonoML is not designed to inherently outperform standard AutoML in stable contexts, i.e. static inflow-data distributions. For this kind of scenario, each incoming instance of data provides decreasing marginal information about the statistics of its source, obeying a law of diminishing returns. If the frequency of inflow data is sufficiently high, then buffering a representative sample becomes cheap enough, in terms of runtime, to annul the headstart advantage of online learning. In these stable contexts, stream-ready AutonoML instead proves its value for rapid-deployment operations, where a valid solution must be provided instantly and in the absence of historical data [187], irrespective of quality. Section 2 supported this idea, proposing that, for a general implementation, data transformations should be initialised immediately within their ML-component shells. In fact, we suggest that, given a pool of ML models/algorithms, a solid AutonoML strategy should bias its initial selections to instance-based incremental learners before exploring more complex batch-based models, thus maintaining respectable predictors while allowing data to accumulate. Such strategies work best when ML-pipeline bundles are part of the design, detailed in Section 8, because one or more ML pipelines can then be developed in parallel to one or more that are currently deployed; the foreground/background partitioning of these operations resembles the page-flip method used in computer graphics. Not unexpectedly, this solution-refinement process is a logical one for user-interactive systems that minimise response latency, e.g. ones where users may wish to steer data exploration, and is employed by the Northstar data-science system within its CASH-solving module, Alpine Meadow [210].

Where the AutonoML paradigm becomes critically necessary is for ML tasks applied within dynamic contexts, specifically scenarios for which ‘concept drift’ occurs [122]. Technically, this refers to the desirable mapping between query-space Q and response-space \mathcal{R} changing over time, e.g. epidemiological notifications being updated to include asymptomatic cases. Notably, real concept drift is often conflated with virtual drift, whereby the data distribution that drives ML-model training changes, e.g. notifications being assessed within a new city. Theoretically, virtual drift does not affect the ideal mapping. Nonetheless, given that an ML model, $M : Q \rightarrow \mathcal{R}$, is already an imperfect approximation of the maximally desirable mapping, we use ‘concept drift’ here loosely; in practice, both forms of drift require ML models to be re-tuned. Thus, if it is to persist usefully in these contexts, an AutonoML system should be able to identify any form of drift and respond appropriately.

On a conceptual level, adaptive strategies can be integrated into an AutoML framework in the manner shown by Fig. 12. This schematic builds upon the CASH-solving system supported by meta-learning, as depicted in Fig. 10, albeit with the additional inclusion of the ML-pipeline bundles

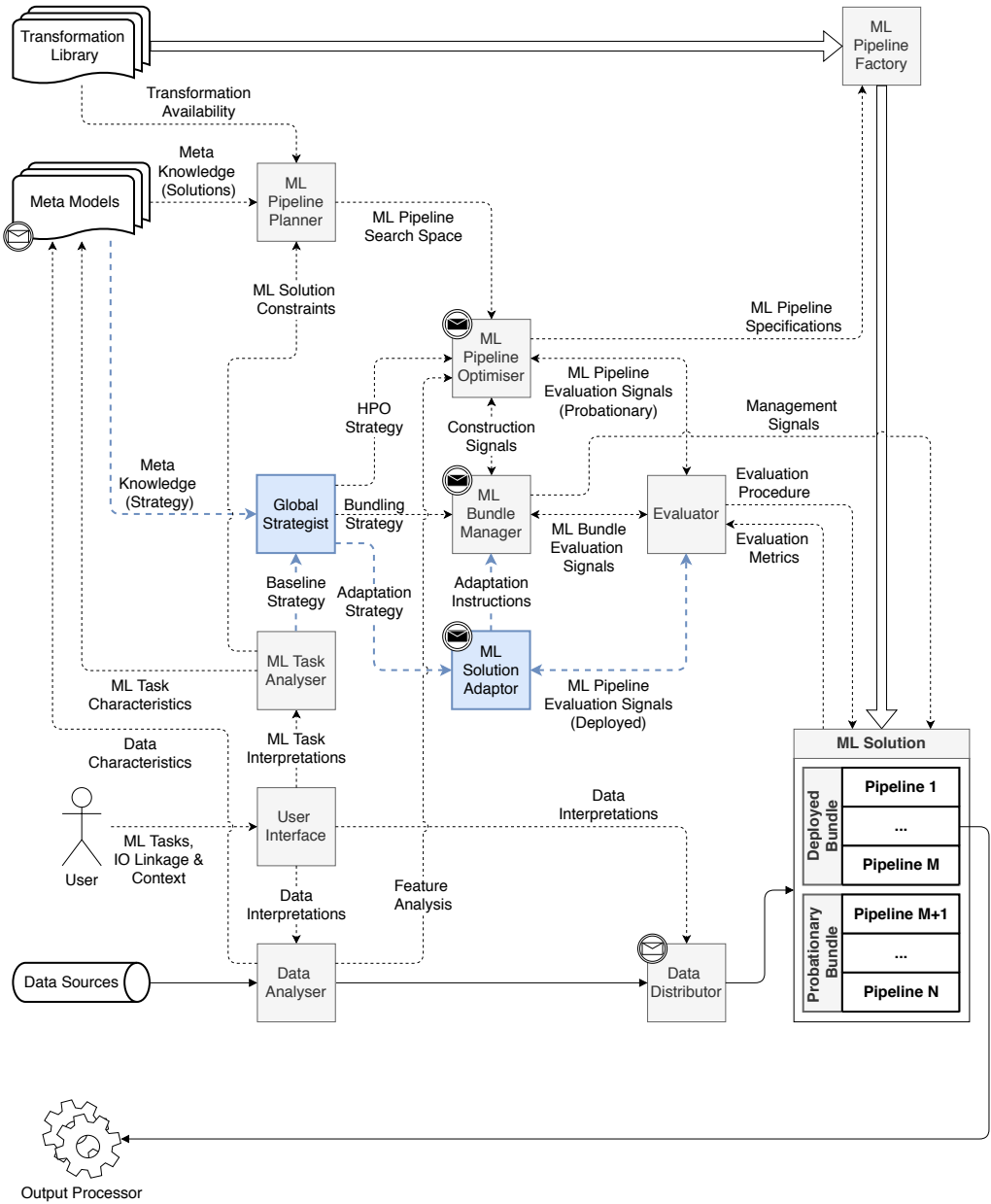


Fig. 12. (Colour online) A high-level schematic of a dynamic AutoML system suited to stream data, where an ensemble of deployed pipelines is monitored for concept drift and reactive strategies are enacted. This architecture is an upgrade to that of Fig. 10, with blue highlights emphasising the main new additions related to adaptation. Dashed arrows depict control and feedback signals. Solid arrows depict dataflow channels. Block arrows depict the transfer of ML components/pipelines. Black/white envelope symbols represent data propagation requests to the data distributor and developmental feedback to the meta-models.

introduced by Fig. 11. Here, just as specialisation required an ML-bundle manager to oversee the ML-pipeline optimiser, a new adaptor module has its own elevated responsibility over the bundle manager. It primarily monitors the performance of individual deployed ML pipelines, polling the evaluator with some frequency over the lifetime of the ML solution, and, when the threshold for drift is passed, it instructs the bundle manager to deal with the failing pipeline accordingly. Based on the employed strategy, this may involve shifting extant ML pipelines between foreground deployment and background probation, not just the outright addition/subtraction of individual learners to/from memory.

As before, this new partitioning of responsibility does require updates to our illustrative architecture. Upon encountering a new ML problem, initial task analysis may suggest a baseline approach, but the global strategist is now responsible for prescribing individual tactics regarding HPO, pipeline ensembling, and adaptation. To reflect that any of these can be improved by prior experience, meta-models now feed their strategic knowledge into the global strategist for further dissemination. In turn, reciprocal meta-model development is extended across both the bundle manager and solution adaptor. As for dataflow control, the data distributor similarly receives signals from all three solution-developing modules to support their activities. Given the potential for undue clutter, Fig. 12 uses shorthand symbols to depict both meta-model feedback and data propagation requests.

Returning to a survey of academic literature, the topic of adaptation has been a thread of ML research for decades. For instance, catastrophic interference has long been a challenge for updatable neural networks [249], in that previously learned knowledge is so easily erased by adjustments to the trained ML model. As a side note, one way to mitigate this effect is in fact to use the specialisation techniques introduced in Section 8, so that adjustments in knowledge are well segregated; this was demonstrated by early adoption of a two-level ensemble within an adaptive monitoring scheme for a water distribution network [118]. Elsewhere, biological inspirations for adaptive classifiers [103] managed to spawn an entire sub-field dedicated to artificial immune systems. Amongst all these investigations, some studies have expressed a recurring principle, that adaptability is the capacity to cope with uncertainty [6]. This view meshes well with the idea of fuzzy logic, where the rigidity of decision boundaries is softened, and so, unsurprisingly, the development of neuro-fuzzy theory has aligned well with researching ML solutions that are robust and easily adaptable. Indeed, fuzzy neural networks have previously been proposed as the basis for evolving intelligent systems [16, 193], and their online-learning algorithms continue to be improved [195].

Of course, most ML algorithms do not have in-built mechanisms that differentiate between contexts. Instance-based incremental learners may evolve as rapidly as possible, closely tracking the receipt of new data, but even these algorithms are typically incapable of controlled non-catastrophic forgetting. One approach for designing exceptions to this rule is to include a domain-encoder in some manner. This is exemplified by a deep-learning architecture that, with some novel gradient-reversal tweaks, branches off a domain-classifier alongside standard label-predicting layers [123]. Loosely expressed, the domain-classifier serves to absorb the influence of differing contexts during training, promoting domain-invariant features for label classification. While this domain classifier arguably internalises only one dimension of concept drift, the more recent Memory-based Parameter Adaptation (MbPA) method [323] works to embed and save previously encountered labelled data, potentially capturing multiple contexts. Parameters for an output network are adjusted based on how embeddings of new instances compare to those that have already been memorised.

From an idealised perspective, the paucity of specialised algorithms is largely irrelevant to AutonoML, which aims to be universally applicable; any strategist in such a framework should employ generic adaptation mechanisms that work irrespective of base learner. Dynamic Weighted Majority (DWM) is one such general method, bundling together an ensemble of experts [204]. If the

ensemble errs, a new expert is added, training only on subsequent data that ideally encapsulates a drifted concept. If an expert errs, adhering too rigidly to outdated concepts, its prediction is given less weight within the ensemble. If confidence in an expert is sufficiently low, it is booted from the ensemble altogether. Another alternative is the Paired Learners (PL) method, which only requires two ML-pipelines [18]. Specifically, a stable learner trains on all the data it encounters after its initialisation, while a reactive learner trains on a moving window of data. If the reactive learner sufficiently outperforms the stable learner, the latter is reset with a new expert. Beyond this, there are many more approaches to adaptation that have been reviewed elsewhere, e.g. in the context of soft sensors [189]. More complex methods include the ‘incremental local-learning soft-sensing algorithm’ (ILLSA) [188] and its batch-based successor, the Simple Adaptive Batch Local Ensemble (SABLE) method [22].

The principle behind SABLE is worth calling particular attention to, as the procedure is actually expressed as a composite of multiple adaptive mechanisms, e.g. batch-based retraining or adding a new expert. The order of these processes is important in optimising predictive accuracy over time [23]. Recent experiments with SABLE have been combined with investigations into batched versions of DWM and PL, similarly decomposed into atomic adaptive mechanisms that could arguably be mixed and matched [24]. Given a strong drive towards integrated adaptive systems [119], this customisation of strategies suits the ethos of *AutonoML*, although, as with the meta-models discussed in Section 7, questions remain as to whether automating such a high-level selection process is tractable.

At the very least, a modular approach for adaptation is appealing; such a philosophy aligns with the design of a general adaptive MCPS architecture, published in 2009 [183], that serves as partial inspiration for the illustrative one developed in this paper. This plug-and-play framework is worth highlighting, as it describes a predictive system operating with constrained ML pipelines/components, employing a meta-learning module and even catering to both multi-objective evaluation and expert knowledge; see Sections 10 and 12, respectively. Most pertinently, it is a system that employs a dynamic and hierarchical aggregation of learners, leveraging lessons from prior research [287–289] that was detailed in Section 8, and has been successfully instantiated several times in the context of soft sensors [184–186]. However, its publication preceded the surge of progress in HPO/CASH over the last decade, resulting in an architecture that is rudimentary in general purpose HPO, while being sophisticated in both automatic adaptation and the integration of multiple relevant ML mechanisms into a coherent framework. It is symbolic of the fact that, while we treat *AutonoML* as a successor to *AutoML*, research and development do not obey convenient orderings.

While devising strategies to maintain ML-model accuracy in shifting contexts is important, there is another key question to consider: when should an adaptive mechanism be triggered? A sustained concept drift, whether real or virtual, is different from an anomaly [60]; any detector should be capable of distinguishing and ignoring the latter. Naturally, there have historically been many approaches to change detection. The Page-Hinkley test is one such example from the 1950s, sequentially analysing shifts in the average of a Gaussian signal [270]. It finds frequent use in adaptive mechanisms, e.g. triggering the pruning of regression-based decision rules based on their errors [11]. On that note, many detection techniques are model-based, assessing concept drift by the deterioration of predictor accuracy. For instance, the Drift Detection Method (DDM) essentially alerts a context change if the error rate of an ML algorithm passes a threshold [121]. The Early Drift Detection Method (EDDM) is more interested in the distance between errors, thus picking up on slower drifts more effectively [19]. The downside with such methods is that they can be computationally expensive, especially in resource-constrained hardware environments. This is particularly true for many non-incremental ML algorithms, which need to re-process historical

data just to ingest a new batch of instances. Alternatively, reminiscent of the filter/wrapper duality of AutoFE, some drift detection methods operate solely on the data, e.g. identifying outliers and shifts based on local densities [277].

Despite all this research into the triggers and processes of adaptation, it is possible to question whether it is too speculative to declare AutonoML as the next step in ML automation. This is certainly open to debate, but the last decade has already teased synergies between the surge in AutoML developments and the theories of dynamic contexts. For instance, while ensemble-based strategies such as SABLE are usually homogeneous in terms of predictor type, HPO studies have considered the feasibility of changing hyperparameters over time [59]. One proposal does this via the online Nelder-Mead algorithm, triggering the search when a Page-Hinkley test identifies concept drift [352]. Then there are efficiency-based investigations tailored for the idea of an MCPS. These include identifying whether segments of a pipeline can be adapted independently [385] and, if so, how these adjustments must be propagated through the rest of an ML pipeline [300]. The interplay of meta-learning and adaptation also remains of interest [7], with the former potentially used to recommend an expert [173] or, at a higher level, an adaptation mechanism.

In any case, it has been the last couple of years that have truly hinted at burgeoning interest in adaptive AutoML. While batch-based and cumulative strategies were evaluated with AutoWeka4MCPS for a chemical-processing case study in 2016 [299], at least three publications have recently been released to push beyond this body of work. One studies the extension of Auto-sklearn to data streams, complete with drift detection [242], while another examines adaptive mechanisms more broadly, leveraging both Auto-sklearn and TPOT [173]. A third works towards a more extensive multi-dataset benchmark, adding H2O AutoML and GAMA to the list, while also using EDDM to identify concept drift and trigger adaptation [57]. Thus, it appears that the transition to AutonoML has already begun. Nonetheless, numerous challenges remain [383]. These include ensuring that adaptive systems are user-friendly, trustworthy, and capable of expert intervention. They must also scale well to big data that is realistic and messy, and, in the long run, work in general settings. In practice, adaptation must also be deployed sparingly, perhaps only when a cost-benefit analysis of an operation identifies a net positive [384]. Intelligent evaluation is key to optimising all aspects of AutonoML.

10 DEFINITIONS OF MODEL QUALITY

Recalling Section 2, the purpose of ML is to learn a model from data that approximates some maximally desirable mapping, $Q \rightarrow \mathcal{R}$. In certain cases, i.e. unsupervised learning and data-exploratory processes, this ground-truth function can seem arbitrarily subjective and vague; if two data scientists are scouring data for ‘interesting’ information, would they agree on their criteria for interest? Would they even know what they are looking for until they find it? Without meta-learning the brain chemistry of a user, or at least the expression of their preferences, an AutoML system cannot estimate the quality of an exploratory ML model until it serves a user-defined purpose elsewhere. Nonetheless, for predictive tasks, e.g. those employing supervised learning, the desired $Q \rightarrow \mathcal{R}$ mapping is much more objective and immediately discernible from inflow data, e.g. hinted at by way of class label. Thus, unsurprisingly, there is a tendency across ML research to focus on the concept of ‘accuracy’. This is understandable, as minimising some metric for the dissimilarity between model and ground-truth function is a universal goal across all ML tasks. However, the automation of ML is ultimately in service of human requirement, and AutonoML, as a field, cannot ignore the many ways by which model quality can be judged.

Conveniently, from a conceptual perspective, broadening the scope of model performance does not require any major changes in the illustrative architecture developed thus far, i.e. Fig. 12. While ML components and ML pipelines have all incorporated an accuracy evaluator, the high-level

evaluator module has been left intentionally abstract from the beginning, linking up with the interfaces of both. There is no reason why the ML-pipeline bundle cannot be fleshed out as a proper wrapper either; Fig. 11 only avoided this for the sake of simplicity. Thus, assuming low-level interfaces have easy access to other model characteristics beyond internally evaluated accuracy, e.g. runtime or memory footprint, higher-level modules are able to optimise the solution according to varied objectives. Admittedly, direct assessment in the Fig. 12 schematic does remain limited to the ML solution, with high-level strategies being only indirectly evaluable. This is likely fine in practice, although, in theory, perhaps high-level strategies should in fact be directly evaluated and optimised; see Section 14.

Notably, when the algorithm-selection problem was first codified in the 1970s, various ways of evaluating models were proposed, including metrics based on complexity and robustness [286]. The vast majority of HPO methods and CASH-solvers over the last decade have disregarded these concepts, although not without cause; error metrics are, on the whole, easier to standardise and calculate across different types of ML model. Even so, the idea that model quality could be based on alternative metrics has persisted throughout the years, with the fusion of multiple criteria likewise studied at least as early as in the 1970s. For instance, despite describing not-for-profit programmes, one publication discussing the ‘efficiency’ of decision making units by combined characteristics [62] is often cited as a forerunner for multi-objective evaluation in KDD systems. Certainly, by the 1990s and the formalisation of KDD, the list of proposed metrics by which to judge an ML-model was broadening, including attributes such as novelty and interpretability, and it remained an open research question as to if and how these could be measured and combined in weighted sum [259].

In recent years, with the advent of HPO methods, sections of the ML and ML-adjacent communities have asserted the need for multi-objective optimisation [160]. A thesis on multi-layer MCPS frameworks explores this topic in depth [5], noting that common approaches may attempt to linearly combine objectives, hierarchically optimise them, or, in the absence of clear priorities, seek solutions that are Pareto-optimal, i.e. where the improvement of one criterion necessarily damages another. In practical terms, the mlrMBO package is one implementation among SMBOs that caters to multiple criteria [40, 161], while, for evolutionary methods, multi-objective genetic algorithms have been well established for many years [78].

Returning to evaluation criteria, typically the first extension researchers explore beyond accuracy is runtime; there is a limit to user patience for any practical ML task, beyond which increased accuracy is not appreciated. Argued to have been largely ignored until the early 2000s [46], the gradual inclusion of runtime into model-evaluation schemes [1, 53, 317, 318, 347] may have mirrored the development of the user-focussed KDD paradigm and the maturation of HPO, e.g. the shift of AutoML from lengthy academic experiments to public-use toolkits. Of course, observing the development time of an ML pipeline and its ML components is trivial, while meta-learning techniques have been proposed to transfer accumulated knowledge to new estimates [172, 244].

Beyond accuracy and runtime, many performance metrics have been proposed for ML solutions, although, to date, consolidative surveys are largely absent. In part, this is because many proposals are only applicable to restrictive subsets of use cases, or have at least been coupled tightly to particular contexts. For instance, FE transformations can be judged based on associated errors and tuning time, but optimising feature-selection percentage, i.e. preferencing fewer features [272, 356], is specific to AutoFE pipeline segments. As another example, an automated learning system for semi-supervised learning (Auto-SSL) uses SVM-reminiscent ‘large margin separation’ as a criterion for HPO, where inflow data is mostly unlabelled [232]. Furthermore, as adaptation research into streaming data and changing contexts continues, it is likely that evolving AutoML systems will require new descriptors for the quality of ML solutions, e.g. metrics relating to temporal stability and robustness.

For now, research strands into ML performance evaluation remain arguably disorganised, at least from the perspective of informing high-level control mechanisms for an AutonoML system. Typical ML benchmarks focus on minimising both loss functions and processing times, which do not necessarily encapsulate the entirety of human requirement. In fact, in seeming response to the age of deep learning and extensively layered DNNs, it is a tenable view that model complexity has recently become a major marker for solution quality, or, more precisely, the lack thereof. Specifically, while the MCPS paradigm has enabled arbitrarily complicated ML solutions to be built, discussed in Section 4, there is an implicit understanding in the ML community that no ML model should be unjustifiably complex. Efforts towards a general meta-learning architecture have espoused this view [176, 177], and the TPOT package demonstrates a practical Pareto-based method of controlling ML-pipeline size [267]. From an ideological perspective, complexity is eschewed by intensifying social desires for explainable/interpretable ML models [132, 293]. Indeed, while the power of ensembling for maximising predictive accuracy is well evidenced, as discussed in Section 8, the process of knowledge distillation into simpler models is partially driven by the opaque nature of these ensembles [100]. However, there are also pragmatic reasons for maintaining simple models, especially when computational resources are in short supply.

11 RESOURCE MANAGEMENT

All sections up to this point have implicitly discussed AutoML and its extension to dynamic contexts, AutonoML, as fundamental developments in data science. The emphasis has been on how novel strategies and theoretical advances in various topics, e.g. optimisation and knowledge transfer, have enabled or upgraded the automation of workflow operations for an ML task, with special focus on designing and maintaining high-quality ML pipelines within a diversity of contexts. However, while the conceptual promise of AutonoML has the potential to revolutionise how society engages with ML on a day-to-day basis, the present reality suggests that practical application is still far behind theory. There may be a proliferation of AutoML packages as of the year 2020, both open-source and commercial, but it is arguable that many lack the application stability to support general uptake. For instance, several benchmarking studies [25, 98, 387] have, between them, encountered process failures for the following: Auto-WEKA, Hyperopt-sklearn [205, 206], Auto-sklearn, auto_ml, TPOT, H2O AutoML, Auto-Tuned Models (ATM) [330], AutoGluon-Tabular, Google Cloud Platform (GCP) Tables, and Sagemaker AutoPilot. This list is virtually guaranteed to not be exhaustive. Thus, it is clear that the challenges remaining are not simply theoretical; AutonoML is a problem of engineering. Managing the automation of ML when subject to resource-constrained hardware is a research thread of its own.

Naturally, it is challenging to proclaim anything too specific on this topic; hardware particulars, including layerings and intercommunications, vary greatly from IOT devices to large-scale servers. However, broadly speaking, an AutonoML system cannot achieve optimal performance without direct awareness of and influence on its hardware environment. An abstraction of these features is illustrated by the use of a resource manager in Fig. 13, a schematic that extends the adaptive AutonoML shown in Fig. 12. In terms of routine operations, this management module is tasked with connecting requests for resources, e.g. memory allocations for data transformations, to the hardware that is available. However, it also represents the opportunity to guide the search for ML solutions, assessing the limitations/benefits of available hardware and evaluating how resources are used in the process of ML activities. With this knowledge, a well-designed resource manager can both constrain solution space directly, e.g. restrict ML-pipeline complexity for low-memory settings, and tailor high-level strategies, e.g. trial DNN candidates only when GPUs are free. This representation also implies that strategies, e.g. adaptive mechanisms, can themselves be adapted over the course of an ML task in response to changing resource circumstances. Overall, most recent

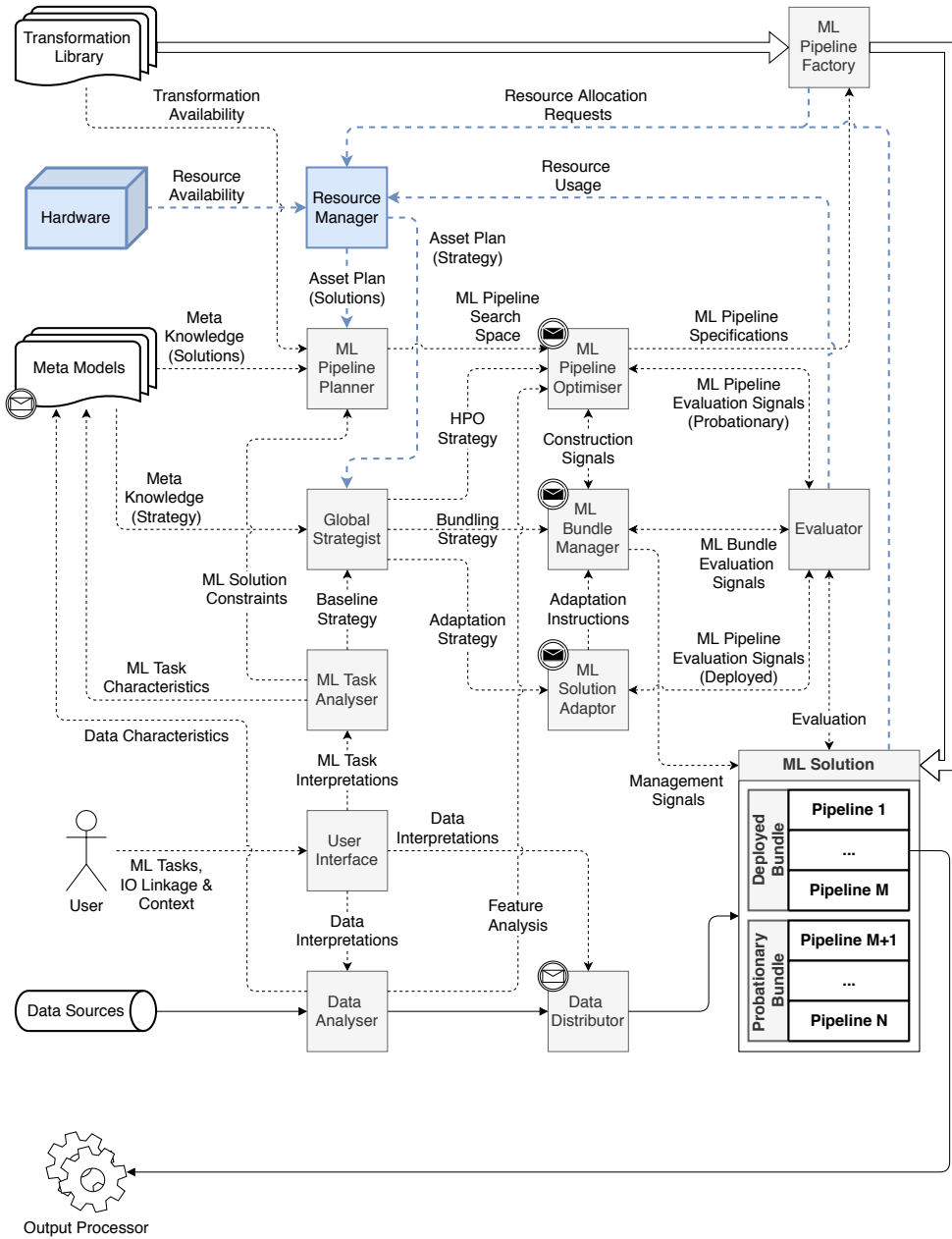


Fig. 13. (Colour online) A high-level schematic of a resource-aware AutoML system, with a manager responsible for asset allocation and strategy. This architecture is an upgrade to that of Fig. 12, with blue highlights emphasising the main new additions related to resource control. Dashed arrows depict control and feedback signals. Solid arrows depict dataflow channels. Block arrows depict the transfer of ML components/pipelines. Black/white envelope symbols represent data propagation requests to the data distributor and developmental feedback to the meta-models.

developments in resource-conscious AutoML can be projected in some way onto this conceptual framework.

Returning to a survey of related research, runtime appears to be the primary focus of efficiency-based studies in AutoML. While Section 10 discussed runtime as a metric for grading ML models/algorithms, resource-management research is more interested in how the system as a whole should function around scarce processing time. In particular, ‘anytime algorithms’ are procedures designed specifically around this concern, providing valid solutions to a problem if interrupted. Incremental ML algorithms are generally most suitable for anytime-compliance, whereas batch-based learning has more to lose if cancelled; this idea is related to the discussion of rapid deployment in Section 9. Importantly, while low-level training processes are good candidates for anytime mechanisms, a robust AutonoML system should be safely interruptible at all levels. As an example, one proposed meta-learning framework acquires its meta-model for ML algorithms offline, but then, upon being applied to a dataset, is able to evaluate an ordered list of recommendations until interrupted, returning the best solution found thus far [138].

Of course, while resilience against interruption is a necessity, efficient operation within time constraints is the ideal. For instance, research into fusing BO with MAB approaches was partially driven by time-budget considerations [101, 159], which is perhaps why some consider BO-HB to be a state-of-the-art CASH-solver [373]. Even so, working within a budget may still be more practically challenging than expected, as, curiously, a recent study found circumstances in which popular AutoML packages, TPOT and H2O AutoML, did not even adhere to user-provided time limits [98]. Other high-level systems have also been the focus of budget-conscious experimentation, with, for instance, the economic principle of return on investment (ROI) used to assess adaptive mechanisms [384]. The ROI in this study retrospectively pits gains in ML-model accuracy against processing time, additionally including several other costs, e.g. random access memory (RAM) and cloud-service communication costs.

Excluding rare exception, the earliest years of modern AutoML research appear to avoid serious discussion about hardware, with efficiency-based developments targeting algorithms in a general manner. However, given the nature of DNNs, the popularisation of NAS has forced developers to be conscious about interfacing with hardware. For instance, the Auto-Keras package automatically limits DNN size based on GPU memory limits [179]. Furthermore, there has been growing interest in designing AutoML systems and packages for clusters of machines, despite the typical challenges of developing distributed ML [89], e.g. designing/using an appropriate ‘programming model’ beyond MapReduce [191]. For some portions of the ML community, this interest appears to be motivated by the needs of time-critical big-data biomedical fields [240], as is exemplified by the application of distributed HPO to a lung-tissue classification problem [306].

Consequently, recent years have witnessed AutoML frameworks being developed for distributed big-data settings, e.g. KeystoneML [320, 322] and ATM [330]. Notably, an early forerunner of this movement is MLbase [211], which is supported by a MAB-based CASH-solver called ‘Training supported Predictive Analytic Queries’ (TuPAQ) [321]. Beyond using a cluster of machines to evaluate ML models in parallel, TuPAQ is also designed for data-parallel scenarios in which a dataset is too large to fit on one machine, thus forcing model updates to rely on the fusion of partial statistics derived from data subsets that are processed on individual worker machines. Accordingly, many of the design choices behind TuPAQ are geared towards optimising against this engineering challenge, such as a mechanism to discard unpromising models before they are fully trained. Such stopping rules are not uncommon, and they are employed, for example, by another cluster-based hyperparameter optimiser named Sherpa [157].

The recent trend of companies providing AutoML as a cloud-based service only complicates matters further. In these settings, an AutoML system must run multiple ML tasks across a pool of

processors in asynchronous fashion. This extension of standard AutoML to multiple devices and multiple tenants (MDMT) requires novel strategies for resource coordination [378]. The cost of data transfers between hardware can become significant in such distributed settings, meaning that hyperparameter configurations must be allocated intelligently between workers when solving CASH. Task schedulers based on runtime estimates have been proposed to support this allocation [332]. Beyond model-parallel and data-parallel approaches, certain frameworks like PipeDream have also explored whether breaking apart an ML pipeline, technically a DNN, may be more efficient, where individual machines are dedicated to the training of individual ML pipeline segments [150].

A notable extension of distributed ML that has gained attention in the last few years is the concept of federated learning, where data samples located on local machines cannot be transferred elsewhere. Because of this restriction, the data may not be independent and identically distributed either. Any associated research has been attributed to a wave of ‘privacy-aware’ ML [132], on account of data scientists only having access to locally trained models and not their underlying data. Recently, the federated-learning community has highlighted a growing need for HPO and other AutoML processes in this context, albeit stressing that communication and computing efficacy for mobile devices cannot be overwhelmed by the traditional objective of model accuracy [190]. Consequently, the year 2020 has witnessed the application of NAS to federated learning, with, for example, the FedNAS framework, which deploys one NAS-solver per local machine and then coordinates them via a central server [151]. Alternatively, the DecNAS framework targets low-resource mobile devices, noting the FedNAS assumption that each local machine is GPU-equipped. In the absence of hardware resources, DecNAS uses subgroups of devices only to train/test candidate models, while the actual searcher is situated at a higher level, i.e. a central cloud [371].

There are further engineering challenges to consider that are often overlooked in theoretical investigations of AutoML. For instance, most benchmarks do not consider inference latency, i.e. the time it takes to convert a query to response. Typically a negligible concern in academic experiments, inference latency can become significant enough in MDMT settings and industrial applications to warrant an automated control system of its own, e.g. InferLine [72], which is responsible for query distribution and provisioning ML bundles. Occasionally, this execution lag can be further diminished by designing and incorporating bespoke hardware, e.g. a Tensor Processing Unit (TPU) [181] or a Neural Processing Unit (NPU) [112]. In this case, an ideal AutoML system should be capable of leveraging optional hardware when appropriate.

On the other end of the spectrum, hardware restrictions can be necessarily severe, such as when designing ML solutions in IOT settings [80]. In this context, an AutoML system must decide whether to centralise its ML model and treat integrated circuits with embedded Wi-Fi as pure sensors, or whether some ML components should be installed on these limited-memory microchips. Research into IOT-based AutoML is presently sparse, but this only serves to suggest that many applications of interest remain for AutoML research.

12 USER INTERACTIVITY

How should artificial intelligence (AI) be designed to interact with humans? This recurring question is inherently profound, but, within the AutoML context, it is a pragmatic one. Traditionally, a core principle behind AutoML is to reduce manual involvement in all ML processes. Indeed, a long-term goal of AutoML is to allow a user to specify a task of interest, hook up relevant data sources, and then relax; the onus is on the system to generate and maintain a high-quality ML solution. However, an ideal framework should also facilitate user participation whenever desired and wherever possible. In fact, a lack of user-friendly control options among contemporary packages may be adversely affecting AutoML uptake altogether [219].

Notably, in some scenarios, human interaction is intrinsic to an ML task; these are generally called human-in-the-loop (HITL). Although this topic is broad, it typically involves a user engaging with an ML system, either by validating query responses, e.g. confirming the classification of an email as spam, or by providing data that best serves the learning process, e.g. deciphering scanned text via ‘Completely Automated Public Turing tests to tell Computers and Humans Apart’ (CAPTCHAs) [354]. The process of an ML system intelligently requesting additional information from an oracle/teacher is called active learning, and it shares faint similarities with the persistent learning and adaptive mechanisms discussed in Section 9. Frequently applied to NLP, studies of active learning have often considered how to optimise the feedback loop between human and AI, e.g. by seeking the most impactful corrections to a model [75] or by accelerating feature extraction between requests [14]. Some efforts have even explored using RL to outright learn active-learning strategies, i.e. tactics for deciding which unlabelled instances of data should form the basis of annotation requests [102]. Essentially, while reliance on human intelligence in HITL settings may seem incompatible with the concept of autonomy, progress towards effectively leveraging this interaction lies firmly within the AutonoML sphere.

More broadly, the design of AutoML systems regularly involves questions about where to link in human control and expertise. Some ML platforms target the ML pipeline, with, for instance, the hybrid Flock system embedding a crowd of users as an FE component [65]. This crowd is asked to nominate and label a feature space for inflow data associated with an ML task, thus encoding attributes that may be challenging for a machine to extract, e.g. the attitude of a person within an image. In theory, though, every control element of an AutonoML system could be influenced by a human expert, as was considered within an adaptive architecture proposed in 2009 [183]. Baking this flexibility into an implementation is an engineering challenge, but it has been suggested that AutoML uptake would be served well by at least designing a few varying levels of autonomy, e.g. user-driven, cruise-control, and autopilot [219]. One recent study has even attempted to systematise which user-driven interactions should be supported; this pool of basic actions was partially drawn from a decomposition of published scientific analyses [137].

Given the expanding complexity of the illustrative framework developed in this paper, we avoid presenting another cumulative depiction of AutonoML. However, the salient elements of advanced user interfacing are shown in Fig. 14. This representation can easily be integrated into resource-aware AutonoML, portrayed in Fig. 13. Crucially, the diagram emphasises the analogies between expert knowledge and meta-knowledge, suggesting that there is no conceptual difference in how prior knowledge is used to tune regular/hyper/system parameters, whether accumulated manually or automatically. In this paper, we have essentially partitioned preparatory high-level ML operations into two objectives: determining a solution search space and determining how to search throughout that space. Within this representation, the analogies are thus reflected by informative signals being directed to both the ML-pipeline planner and global strategist. This is a consistent design choice; Section 11 suggested that recommendations based on the practicalities of resource availability should be likewise distributed. Beyond all that, Fig. 14 also portrays active-learning processes, with model evaluation exposed to the user. User response may be to tweak either the model or how a new model is found, but data sources may also be supplemented or adjusted as a reaction.

Regardless of design, if this growing focus on interactivity appears to contradict the principles of automation, it is explained by the following: concurrently to the field of AutoML surging in prominence, the last decade has also witnessed a parallel movement towards AI accountability [132, 293]. A popular idea is that an AutonoML system can be treated as a black box, but, if required, its processes should also be completely observable and, ideally, comprehensible. For instance, the Automatic Statistician project merges the philosophies of AutoML and explainable AI by

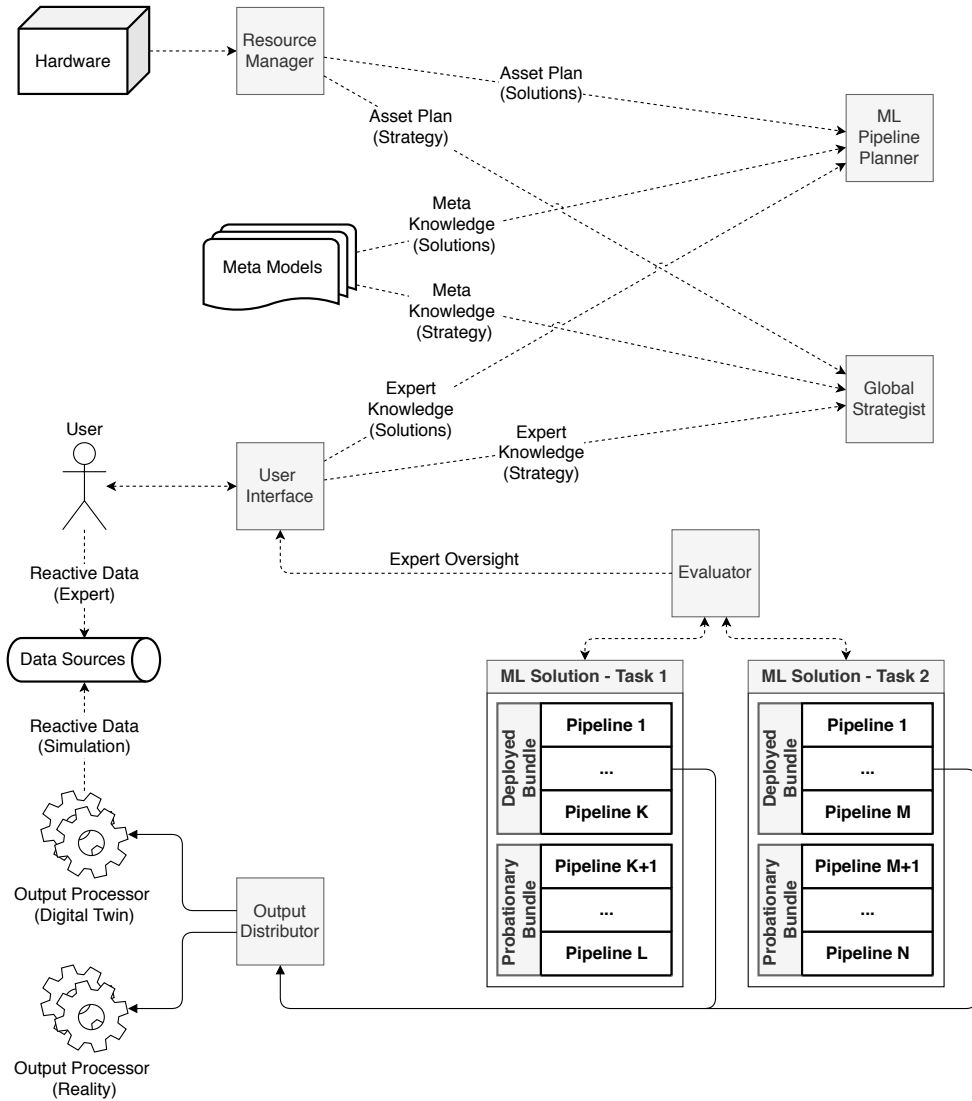


Fig. 14. A high-level schematic of expert-knowledge and digital-twinning mechanisms in an AutoML system. These upgrades should be applied in the architectural context of Fig. 13. Dashed arrows depict control and feedback signals. Solid arrows depict dataflow channels.

aiming to generate human-readable reports from both data and models [324]. Moreover, the idea of interpretability is appealing even during the development of an ML solution; a recent clinician-supported study suggests that predictive performance is more effectively fine-tuned by a user if that user assesses/augments the internal human-understandable rules of a model rather than individual instances of inflow data [133].

Of course, whether as a black box or with its machinery exposed, facilitating ease-of-use for an AutoML system, thereby encouraging lay usage, brings new risks into focus. Specifically, there are intensifying calls to safeguard users from poor or even dangerous conclusions arising

from data-science illiteracy [42]. Northstar, an AutoML system motivated by user interactivity and democratisation, exemplifies consideration of these topics by proposing a module to Quantify the Uncertainty in Data Exploration (QUDE), which acts as a sophisticated error-detection mechanism for user interactions [210]. Ultimately, it is evident that, if AutonoML is to be successfully integrated into society, the design of a UI will need just as much attention as any other subsystem.

13 TOWARDS GENERAL INTELLIGENCE

Over the course of many sections, numerous practical challenges have been raised with respect to the automation of ML. Most of these stem from limitations in the resources that are presently available. Without some technological paradigm shift, e.g. quantum computing [36], searching for a data-transformation pipeline that suits some ML task, particularly a predictive one, will continue to be typically measured in hours, if not by longer timescales. Nonetheless, with research continuing to explore and develop this space, evaluating AutoML packages by the quality of their solutions, i.e. any metric discussed in Section 10, may eventually become less compelling. In the long run, the utility of AutonoML may be determined by another factor: generalisability.

By and large, most AutoML packages have not been developed with flexible data inflow in mind. With many data transformations expecting inputs of a consistent size and shape, even ingesting arbitrary images was considered a significant deep-learning advance, with spatial pyramid pooling promoted as an effective alternative to substandard cropping/warping processes [152]. Thus, unsurprisingly, extensive manipulations of heterogeneous data do not appear particularly common in ML. By the time disparate data passes through the pre-processing segments of an ML-pipeline, it has usually been imputed/concatenated into a uniform vector space for the remaining FE and predictive segments. The approach is similar even when dealing with time series; several chemical-processing studies exemplify this by synchronising and combining data streams from various sensors into vectorised snapshots of an industrial environment [183, 298]. In fairness, this can occasionally be a design choice, as these same studies do actually involve frameworks capable of constructing ML-pipeline bundles and partitioning inflow data in sophisticated ways [297], theoretically enabling the delegation of individual ML experts to distinct sensors.

Regardless of implementation, it is clear that being able to generalise inflow data extends the power of AutonoML, especially in IOT settings [80]. There is a growing appreciation of this fact, with, for example, one recently proposed AutoML system promoting its relatively novel ability to ingest different kinds of data, including tables, text, images, and audio [137]. This is why, from the very beginning in Section 2, the approach in this paper towards conceptualising AutonoML has emphasised the need to treat inflow data as generally as possible.

In the long term, though, the ideal AutonoML system is one that does not just cater to generic data inflow; it is an application that is flexible in the types of tasks it can handle and, as Fig. 14 suggests, ideally able to operate several at the same time. This means extending beyond the supervised learning that a vast majority of present AutoML packages focus on. This also means considering how to contextualise novel ML operations beyond model selection as they arise. Section 8 has already mentioned knowledge distillation as a potential task worthy of its own module. Another example is shown in Fig. 14, which attempts to represent one manifestation of the concept recently termed ‘digital twinning’, heralded at least as early as in the 1990s under the speculative label of ‘mirror worlds’ [128], whereby high-fidelity simulations of physical systems act as proxies for reality. In this schematic depiction, ML solutions can be fed consecutive data that is simulated as a reaction to previous model outputs, where, for various reasons, the acquisition of such data may be infeasible in the real world.

Now, granted, there have been substantial efforts to automate solution search within less typical ML contexts, particularly recently. For instance, one AutoML framework specifically targets semi-supervised learning [232]. Another one applies the principles of NAS to the trending topic of generative adversarial networks (GANs) [140], with related work attempting to automatically distill a GAN [113]. However, most of these efforts currently appear to be proof-of-principle, constructed in bespoke manner for the novel contexts, as opposed to extending existing capabilities. Of course, there is a valid discussion to be had whether attempting to develop a one-size-fits-all system is the right functional approach, but it must also be acknowledged that the learning machine AutoML is attempting to support, i.e. the brain of a data scientist, is itself capable of grappling with a diversity of learning tasks, regardless of human efficiency.

Ultimately, a general-purpose AutoML implementation may not be as blue-sky as one might expect. In the years prior to the optimisation community pushing AutoML, KDD researchers dwelled significantly upon ontologies of ML tasks while devising recommender systems. In essence, mappings were studied between ML tasks and the structure of ML pipelines [105]. While these notions of hard constraints have trickled into the AutoML sphere, the technical achievements behind CASH have yet to be fully explored against the wide-ranging KDD paradigm. It is conceivable that, with good design of a specification language, an AutoML system could accommodate a variety of ML task types, each of which would be translated into ML-pipeline structural constraints by a KDD approach and then fleshed out by a CASH solver. Naturally, various tasks may require unique ML components to be imported from distinct ML libraries, but this is not an obstacle; it is possible to mesh codebases even for disparate programming languages. One recent approach to this wrapped up environment-specific ML components as web services [256, 257]. Furthermore, with robust mechanisms for persistent learning and the long-term accumulation of meta-knowledge, there is potential for long-life AutoML to become even more adaptive within unfamiliar contexts, leveraging associations between the meta-features of an ML task and its optimal solution.

Admittedly, the major caveat to the AutoML paradigm as it currently stands is that it can only evolve via combinatorial exploration within pre-coded spaces of ML components, adaptive mechanisms, ML-task types, and so on. This means that the extended generality of a framework like AutoCompete [335], designed to tackle arbitrary machine learning competitions, remains subject to the hard limit of ML scenarios that the developers either have experienced or can imagine. Nonetheless, and this is no foreign idea to the ML community [233], future developments in the field of AutoML/AutoML may well have a bearing on the quest for artificial general intelligence (AGI).

14 DISCUSSION

Each section of this review has honed in on a particular facet of ML, primarily high-level, surveying the benefits/challenges that its automation may produce/encounter in the pursuit of maximally desirable data-driven models. However, unlike most contemporary reviews of AutoML, there has been an intended emphasis on integration woven throughout this work, with a concurrently designed conceptual illustration demonstrating one way in which to blend these ML mechanisms together. Unsurprisingly, there is more to discuss regarding the big picture of integration.

The overwhelming majority of existing large-scale AutoML frameworks, if not all, are designed prescriptively from above. In essence, developers select phenomena they wish to see in the system, e.g. HPO and meta-learning, and then build around these, often maintaining some sort of modularity. Inevitably, this approach necessitates design decisions, either implicitly or explicitly, that prompt a debate about encapsulation and intercommunication. As an example, one may question whether Fig. 5 actually serves as an authoritative blueprint for a CASH-solving AutoML. For instance, is the separation between ML-solution planner and optimiser overkill when an MCPS is not involved?

Should not the evaluator be responsible for some data propagation requests, rather than leaving them all to the optimiser? Perhaps instantiating an IDC per ML component is too large of a footprint, and the onus of intra-component data distribution should also be on the high-level distributor? Ultimately, pragmatics will determine implementational specifics, and a Unified Modeling Language (UML) diagram of a resulting AutoML codebase may look dramatically different from the networked abstractions in Fig. 5.

On the other hand, each schematic in this work has been presented as a broad conceptual depiction of ML/AutoML/AutonoML operations; the demonstrative power of a schematic depends simply on whether it captures the salient features of the associated mechanisms. By this metric, Fig. 5 is not expected to be controversial in its depiction of solving CASH, and any legitimate tweaks to the diagram must maintain the same expression of that process. That raises the question: design choices aside, do all the diagrams in this review definitively reflect the appropriate application of an AutoML mechanism? In truth, and this is why the large-scale integration of AutonoML is deserving of research attention, probably not. The optimal interplay of processes is unknown.

To elaborate on this, it is first worth noting that a fundamental criticism of AutoML is that automation merely shifts the level at which human involvement is required [233]. Base-level ML algorithms select the parameters of a model. CASH-solvers select the parameters of an ML algorithm, i.e. hyperparameters. So then, what selects the parameters of a CASH-solver, i.e. hyper-hyperparameters? Certainly, each time this search is elevated by a level, the time required to solve an ML problem balloons out; it may presently be infeasible to optimise the high-level optimisers. However, relying on user-defined system parameters could face the same negative criticism earned by the use of default model parameters [20]. There may be an eventual need to discuss telescoping optimisations and thresholds for diminishing utility.

Crucially, matters are complicated when multiple methodologies interact. The illustrative framework developed in this paper has been grounded in the debatable assumption that HPO, specialisation and adaptation are best enacted on the same level above the ML model. Indeed, there is a simple logic to the equal treatment of associated strategies that is depicted in Fig. 12. Yet, perhaps the parameters of specialisation/adaptation strategies should be optimised. Perhaps HPO/adaptation strategies should operate differently across a bundle of ML pipelines, i.e. be specialised. Alternatively, perhaps HPO/bundle strategies should be adapted for concept drift. At the moment, beyond a disinclination for complex architecture, there is no systematic way to determine when any of these arrangements are worthwhile or not, especially when benchmarking AutoML remains in its infancy. If, in fact, an AutonoML framework does turn out to involve sophisticated multi-level networks of mechanisms in order to perform optimally, the field will begin to overlap with the area of complex systems, insinuating possibilities regarding emergent behaviour and the process of learning. We defer further speculation on the topic.

Importantly, it must be acknowledged that, irrespective of engineering, certain trade-offs are likely to remain constants in the design of an ML/AutoML/AutonoML system. These include:

- Speed versus accuracy – Low runtime and low loss are archetypal competing objectives, the contest of which, given the computationally expensive loop of model/algorithm probation for CASH, will remain a challenging disincentive for layperson uptake. Having said that, the incredible improvement in computational power, memory and availability of large amounts of data means that the computationally hungry AutoML research emerging in recent years has become tractable, and the speed with which ML models of equivalent accuracy can be trained has also been dramatically increased over recent years. Naturally, there is an ongoing need for research in this space to take advantage of inevitably improving computational power.

- Stability versus plasticity – Highlighted by the extremes of entrenchment and catastrophic forgetting [249], any adaptive system faces this dilemma when seeking an equilibrium between learning new concepts and maintaining extant knowledge. While this classical dilemma related to learning systems is unlikely to go away, it has been ameliorated by ongoing progress in the thorough and continuous validation of both acquired knowledge and the relevance of deployed ML models.
- Control versus security – Given an AutoML package, broadening the scope of user access and influence is known to improve model performance for a human expert. On the other hand, the opposing result holds for an amateur [376]. This emphasises just how important the AutoML/AutonoML research facet of user interactivity is, as discussed in Section 12, along with, more broadly, the debate around HITL challenges for AI systems in general. These have not yet been addressed in any significant way.
- Generality versus simplicity – The capacity for one system to tackle any ML task is appealing. However, without solid design principles to manage model complexity and feature bloat, the associated codebase can suffer from being too heavyweight for bespoke contexts. These competing attributes are also related to the tension between the accuracy afforded by complexity and the requirement for models to be both transparent and interpretable. In essence, there is a need for progress in both directions. For instance, flexible multi-objective optimisation approaches are expected to extend AutoML/AutonoML beyond a communally prevalent focus on predictive accuracy, enhanced in recent years by additional considerations of model/algorithmic complexity and both runtime and memory usage, but these approaches must also be robust and easily applicable.

Unsurprisingly, finding the right balance for each of the above trade-offs is a significant research endeavour in its own right.

As a final comment on integration, it may end up that the ‘prescribed-from-above’ approach proves entirely inappropriate for AutonoML systems of the future. Admittedly, there is an appeal to designing an AutonoML framework in a modular manner, insofar as the operation of all desirable mechanisms is clearly demarcated and interpretable, but, as DNNs have exemplified, sometimes optimal performance is hidden within obtuse arrangements of granular base elements. If ML algorithms can be designed from scratch by evolutionary means [282], then perhaps an AutonoML architecture itself can and should be grown from first principles, where desired mechanisms arise organically from the pursuit of well-designed objectives. For now, though, this remains a theoretical curiosity, with pragmatism likely to continue pushing the prevalent top-down approach of AutoML design. Biomimetics may be an appealing direction for AutonoML [6], but fashioning a tool remains easier than growing a brain.

Regardless of how the ML community decides to approach the architectural integration, it seems clear that, in the long term, AutonoML has the potential to revolutionise the way that ML is practiced, with significant implications for everyday decision-making across diverse facets of the human experience. That said, history has also witnessed at least two AI winters, primarily on the back of unmet expectations, and there is debate whether the limitations of deep learning are severe enough to herald another in the near future [308]. Thus, while crucial advances in HPO, meta-learning, NAS, adaptation, and so on, are arguably academically impressive, the evolution of the field is not served by overhype.

Therefore, a standard question to ask is this: has the automation of ML had an impact on society? This cannot be answered of AutonoML, as, with the exception of early pioneering work on this subject in 2009–2010 [183, 188], ML-model search in dynamic contexts has only begun garnering broader attention within the last year or so [57, 173, 242]. However, CASH was formalised in

2013 [338], meaning that the modern notion of AutoML has now had a while to diffuse through the data-science community and the mainstream. Certainly, there has been a recent proliferation of both GitHub repositories and commercial websites touting AutoML services.

As of 2020, the answer to the question on impact remains inconclusive. Notably, while it is expected that any publication proposing a new AutoML implementation or subsystem will promote its successes wherever evident, there are likewise strong indications that automation does not intrinsically improve ML outcomes. This has been the case from the beginning, with a 2013 graduate-school project attempting to apply Auto-WEKA to a Kaggle competition, assuming the perspective of a layperson, which eventually reached the following conclusion: “The automated procedure is better than the average human performance, but we do not achieve performance near the top of the ladder, and we cannot even compete with the single default model trained on all the data.” [17]

Mixed sentiments appear to have persisted to date, reoccurring within the few independent evaluations of AutoML packages that have been made, even when the results can be spun one way or another. An example is provided by a 2020 benchmark [149], which uses the performance on several OpenML datasets to assess the following four systems: Auto-sklearn, TPOT, H2O AutoML, and AutoGluon. The study positively claims that at least one package performs equal to or better than humans for seven out of twelve datasets. However, upon excluding three ML tasks where humans and almost all tested packages are capable of perfect accuracy, the results can also be interpreted with equal validity to imply that AutoML systems fail to outperform humans in five, almost six, out of nine contexts.

Conclusions are likewise sparse when solely comparing AutoML packages against each other rather than against human practitioners. One benchmarking study of seven open-source systems notes variations in performance, but it also establishes that there is no perfect tool that dominates all others on a plurality of ML tasks [339]. Proprietary systems are much more difficult to evaluate, however a 2017 investigation into ML-service providers is likewise unable to identify an outlier [376]. Returning to open-source packages, another comparison of four major systems across thirty-nine classification tasks [135] found that, after four hours of operation, all could be outperformed by a random forest for certain problems. The packages also struggled with datasets involving high dimensionality or numerous classes.

Accordingly, scepticism is warranted with any declaration of a state-of-the-art system. There are numerous deficiencies in academic culture, with, for instance, publish-or-perish pressures occasionally leading to shortcuts being taken when making research claims, the generality of which is sometimes unsupported [132]. Computer science is also a field that moves fast, and publications are often released without adequate information that would allow results to be easily reproduced [274].

In fairness, though, AutoML appears most burdened by the fact that there does not yet seem to be a consensus about where its greatest advantages lie. Of the limited independent benchmarks that exist, most adhere to traditional evaluations of predictive accuracy after set amounts of processing time. However, given that a distribution of models engineered by a large crowd of humans on an ML task is likely to have a strong maximal accuracy, perhaps the role of AutoML is one of efficiency instead, i.e. achieving a reasonable result in a short time. If so, such an assessment would mirror conclusions drawn about easy-to-use off-the-shelf forecasting algorithms and their utility for non-experts [224]. Passing judgement is further complicated by the fact that an AutoML/AutonoML system effectively involves the interplay of many high-level concepts, some that may not be shared with other packages. Depending on implementation, these subsystems and constraints may also be difficult if not impossible to turn off or modify, respectively. One may then ask the following: on what basis should a one-component system with a warm-start function be compared against an MCPS without meta-learning? In short, rigorously comprehensive benchmarking is

substantially less developed than the state of AutoML engineering, making it challenging to verify that any theoretical claims propagate into real-world problems. Even ablation studies, appropriate for examining such complex systems, are rare enough to stand out when they arise [98].

The upshot is that, if AutoML is to truly become a disruptive technology, there is more work to be done beyond just the technical front. Indeed, a recent study of the online OpenML platform shows that less than 2% of its users and workflows have adopted associated techniques [219]. Some hints of why this is the case are present in interviews with data scientists, which reveal that, while time-savings resulting from automation are appreciated, there remain several negative sentiments around AutoML [355]. Concerns include the loss of technical depth for operators, a focus on performance rather than insight, the inability to infer domain knowledge, and issues of trust. The authors of the study concluded that, in general, data scientists are open to their work being augmented, not automated, by AI. Likewise, a deeper study into trust has suggested that users may be more open to accepting AutoML if visualisations are prioritised for input-data statistics, FE processes and final model evaluations, although, curiously, the sampled data scientists were less concerned about the internals of the learning process, including HPO [87]. Of course, these are but preliminary perspectives from the academic side. An accurate picture of social engagement with AutoML, along with recommendations to promote it, will require a much broader review of application and uptake.

15 CONCLUSIONS

In this paper, we have surveyed a series of research threads related to AutoML, both well-established and prospective. In so doing, we have identified both benefits and obstacles that may arise from integrating them into one centralised architecture.

To support this review, we have also steadily built up a generic conceptual framework that illustrates how this integration may be achieved, all with the aim of supporting an effective data-driven learning system that needs minimal human interaction. The following is a summary of this process:

- Section 2 introduced a fundamental ‘ML component’, attempting to encapsulate a superset of low-level processes applicable to any ML task, e.g. the data-driven algorithmic development of a model.
- Section 3 schematised a component-selection mechanism, identifying that the search for an optimal ML model/algorithm and associated hyperparameters is the core goal of AutoML. This section reviewed high-level optimisation strategies.
- Section 4 upgraded the framework to revolve around an ‘ML pipeline’, acknowledging that an ML model is more accurately described as a composition of transformations. This section reviewed the management of multi-component ML solutions.
- Section 5 described NAS as a neural-network specialisation of the multi-component framework detailed thus far. This section reviewed NAS.
- Section 6 considered the segmentation of ML pipelines, specifically focussing on optimising pre-processing transformations. This section reviewed AutoFE.
- Section 7 schematised a meta-learning mechanism, designed to incorporate knowledge from previous ML experiments. This section reviewed the leveraging of meta-knowledge.
- Section 8 upgraded the framework to revolve around ‘ML-pipeline bundles’, enabling high-level model combinations and, eventually, the ability to juggle ML pipelines between development and deployment. This section reviewed the management of ensembles.

- Section 9 upgraded the framework to strategise for dynamic contexts, defining persistent learning and adaptation as core prerequisites for AutonoML. This section reviewed managing both streamed data and concept drift.
- Section 10 elaborated the evaluation of an ML pipeline and associated mechanisms. This section reviewed metrics for model quality.
- Section 11 considered framework improvements to manage available hardware. This section reviewed automating operations subject to limited resources.
- Section 12 considered framework improvements to facilitate the inclusion of expert knowledge and user control. This section reviewed human-AI interactivity.
- Section 13 emphasised the flexibility of the framework to handle both arbitrary inflow data and ML tasks. This section reviewed attempts to generalise AutoML.

Finally, Section 14 discussed AutoML/AutonoML from two angles: the overarching challenges of integration and the current state of mainstream engagement.

As this work is primarily intended to be a review of associated research fields, we have had to keep the depth of technical discussions balanced. Accordingly, while the illustrative architecture presented in this work is broadly encompassing and logically motivated, it should not necessarily be seen as an authoritative blueprint for an automated learning system; it is but a preliminary attempt to synthesise numerous topics that have not necessarily been previously combined.

Ultimately, our goal has simply been to show that AutonoML is, in principle, a feasible pursuit as of the current year, provided that a diverse array of concepts are identified and intelligently integrated. Now, granted, the field of AutoML, like deep learning, is thriving on its own in the modern era, especially with the availability of computational power, memory, and data. However, identifying and synthesising these concepts are not trivial; in our view, AutoML cannot properly evolve towards AutonoML without much deeper thought on this topic. The true challenges of venturing into the realm of complex adaptive systems, specifically through the development/emergence of an AutonoML framework as a living and perpetually evolving entity, still lie ahead. This paper is intended to stimulate discussion on what is required and how we should approach such an endeavour.

REFERENCES

- [1] Salisu Abdulrahman. 2017. *Improving Algorithm Selection Methods using Meta-Learning by Considering Accuracy and Run Time*. Ph.D. Dissertation. Faculdade de Economia da Universidade do Porto.
- [2] Salisu Mamman Abdulrahman, Alhassan Adamu, Yazid Ado Ibrahim, and Akilu Rilwan Muhammad. 2017. An Overview of the Algorithm Selection Problem. *International Journal of Computer (IJC)* 26, 1 (2017), 71–98.
- [3] T. Agasiev and A. Karpenko. 2017. The Program System for Automated Parameter Tuning of Optimization Algorithms. *Procedia Computer Science* 103 (2017), 347–354. <https://doi.org/10.1016/j.procs.2017.01.120>
- [4] David W. Aha. 1992. Generalizing from Case Studies: A Case Study. In *Machine Learning Proceedings 1992*. Elsevier, 1–10. <https://doi.org/10.1016/b978-1-55860-247-2.50006-1>
- [5] Bassma Al-Jubouri. 2018. *Multi-criteria optimisation for complex learning prediction systems*. Ph.D. Dissertation. Bournemouth University.
- [6] Frédéric Alexandre. 2016. Beyond Machine Learning: Autonomous Learning. In *Proceedings of the 8th International Joint Conference on Computational Intelligence*. SCITEPRESS - Science and Technology Publications. <https://doi.org/10.5220/0006090300970101>
- [7] Abbas Raza Ali, Marcin Budka, and Bogdan Gabrys. 2015. *A Review of Meta-level Learning in the Context of Multi-component, Multi-level Evolving Prediction Systems*. Technical Report. Bournemouth University. <https://doi.org/10.13140/RG.2.2.32797.54243>
- [8] Abbas Raza Ali, Marcin Budka, and Bogdan Gabrys. 2019. Towards Meta-learning of Deep Architectures for Efficient Domain Adaptation. In *PRICAI 2019: Trends in Artificial Intelligence*. Springer International Publishing, 66–79. https://doi.org/10.1007/978-3-030-29911-8_6
- [9] Abbas Raza Ali, Bogdan Gabrys, and Marcin Budka. 2018. Cross-domain Meta-learning for Time-series Forecasting. *Procedia Computer Science* 126 (2018), 9–18. <https://doi.org/10.1016/j.procs.2018.07.204>

- [10] Shawkat Ali and Kate A. Smith-Miles. 2006. A meta-learning approach to automatic kernel selection for support vector machines. *Neurocomputing* 70, 1-3 (dec 2006), 173–186. <https://doi.org/10.1016/j.neucom.2006.03.004>
- [11] Ezilda Almeida, Carlos Ferreira, and João Gama. 2013. Adaptive Model Rules from Data Streams. In *Advanced Information Systems Engineering*. Springer Berlin Heidelberg, 480–492. https://doi.org/10.1007/978-3-642-40988-2_31
- [12] Alec Anderson, Sebastien Dubois, Alfredo Cuesta-infante, and Kalyan Veeramachaneni. 2017. Sample, Estimate, Tune: Scaling Bayesian Auto-Tuning of Data Science Pipelines. In *2017 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE. <https://doi.org/10.1109/dsaa.2017.82>
- [13] Alec Wayne Anderson. 2017. *Deep Mining: scaling Bayesian auto-tuning of data science pipelines*. Ph.D. Dissertation. Massachusetts Institute of Technology.
- [14] Michael R. Anderson, Dolan Antenucci, and Michael J. Cafarella. 2016. Runtime Support for Human-in-the-Loop Feature Engineering Systems. *IEEE Data Eng. Bull.* 39 (2016), 62–84.
- [15] Marcin Andrychowicz, Misha Denil, Sergio Gómez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. 2016. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Eds.), Vol. 29. Curran Associates, Inc., 3981–3989. <https://proceedings.neurips.cc/paper/2016/file/fb87582825f9d28a8d42c5e5e58b23d-Paper.pdf>
- [16] Plamen Angelov and Nikola Kasabov. 2005. Evolving computational intelligence systems. In *Proceedings of the 1st international workshop on genetic fuzzy systems*. 76–82.
- [17] Anonymous. 2013. Kaggle vs. Auto-WEKA. (2013). <https://www.cs.ubc.ca/~nando/540-2013/projects/p21.pdf> Source: <https://www.cs.ubc.ca/~nando/540-2013/projects/p21.pdf>
- [18] Stephen H. Bach and Marcus A. Maloof. 2008. Paired Learners for Concept Drift. In *2008 Eighth IEEE International Conference on Data Mining*. IEEE. <https://doi.org/10.1109/icdm.2008.119>
- [19] Manuel Baena-Garcia, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, R Gavalda, and R Morales-Bueno. 2006. Early drift detection method. In *Fourth international workshop on knowledge discovery from data streams*, Vol. 6. 77–86.
- [20] Anthony Bagnall and Gavin C. Cawley. 2017. On the Use of Default Parameter Settings in the Empirical Evaluation of Classification Algorithms. *ArXiv* (March 2017). arXiv:cs.LG/1703.06777v1
- [21] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2017. Designing Neural Network Architectures using Reinforcement Learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. <https://openreview.net/forum?id=S1c2cvqee>
- [22] Rashid Bakirov, Bogdan Gabrys, and Damien Fay. 2015. On sequences of different adaptive mechanisms in non-stationary regression problems. In *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE. <https://doi.org/10.1109/ijcnn.2015.7280779>
- [23] Rashid Bakirov, Bogdan Gabrys, and Damien Fay. 2017. Multiple adaptive mechanisms for data-driven soft sensors. *Computers & Chemical Engineering* 96 (jan 2017), 42–54. <https://doi.org/10.1016/j.compchemeng.2016.08.017>
- [24] Rashid Bakirov, Bogdan Gabrys, and Damien Fay. 2018. Generic adaptation strategies for automated machine learning. *ArXiv* (December 2018). arXiv:cs.LG/1812.10793v2
- [25] Adithya Balaji and Alexander Allen. 2018. Benchmarking Automatic Machine Learning Frameworks. *ArXiv* (August 2018). arXiv:cs.LG/1808.06492v1
- [26] Rémi Bardenet, Máttyás Brendel, Balázs Kégl, and Michele Sebag. 2013. Collaborative hyperparameter tuning. In *International conference on machine learning*. 199–207.
- [27] Enrique Barreiro, Cristian R. Munteanu, Maykel Cruz-Monteagudo, Alejandro Pazos, and Humbert González-Díaz. 2018. Net-Net Auto Machine Learning (AutoML) Prediction of Complex Ecosystems. *Scientific Reports* 8, 1 (aug 2018). <https://doi.org/10.1038/s41598-018-30637-w>
- [28] Yoshua Bengio. 2000. Gradient-Based Optimization of Hyperparameters. *Neural Computation* 12, 8 (aug 2000), 1889–1900. <https://doi.org/10.1162/089976600300015187>
- [29] Kristin P. Bennett, Gautam Kunapuli, Jing Hu, and Jong-Shi Pang. 2008. Bilevel Optimization and Machine Learning. In *IEEE World Congress on Computational Intelligence*. Springer, Springer Berlin Heidelberg, 25–47. https://doi.org/10.1007/978-3-540-68860-0_2
- [30] Hilan Bensusan. 1998. God doesn't always shave with Occam's razor — Learning when and how to prune. In *Machine Learning: ECML-98*. Springer Berlin Heidelberg, 119–124. <https://doi.org/10.1007/bfb0026680>
- [31] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of machine learning research* 13, Feb (2012), 281–305.
- [32] James Bergstra, Brent Komer, Chris Eliasmith, and David Warde-Farley. 2014. Preliminary evaluation of hyperopt algorithms on HPOLib. In *ICML workshop on AutoML*.
- [33] James Bergstra, Dan Yamins, and David Cox. 2013. Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms. In *Proceedings of the 12th Python in Science Conference*. SciPy. <https://doi.org/10.25080/majora-8b375195-003>

- [34] 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.
- [35] Róger Bermúdez-Chacón, Gaston H. Gonnet, and Kevin Smith. 2015. *Automatic problem-specific hyperparameter optimization and model selection for supervised machine learning*. Technical Report. Zürich. <https://doi.org/10.3929/ethz-a-010558061>
- [36] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. 2017. Quantum machine learning. *Nature* 549, 7671 (sep 2017), 195–202. <https://doi.org/10.1038/nature23474>
- [37] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. 2010. MOA: Massive online analysis. *Journal of Machine Learning Research* 11, May (2010), 1601–1604.
- [38] Besim Bilalli. 2018. *Learning the impact of data pre-processing in data analysis*. Ph.D. Dissertation. Universitat Politècnica de Catalunya.
- [39] Besim Bilalli, Alberto Abelló, Tomàs Aluja-Banet, Rana Faisal Munir, and Robert Wrembel. 2018. PRESISTANT: Data Pre-processing Assistant. In *International Conference on Advanced Information Systems Engineering*. Springer International Publishing, 57–65. https://doi.org/10.1007/978-3-319-92901-9_6
- [40] Bernd Bischl, Jakob Richter, Jakob Bossek, Daniel Horn, Janek Thomas, and Michel Lang. 2017. mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions. *ArXiv* (March 2017). arXiv:stat.ML/1703.03373v3
- [41] Avrim L. Blum and Pat Langley. 1997. Selection of relevant features and examples in machine learning. *Artificial Intelligence* 97, 1-2 (dec 1997), 245–271. [https://doi.org/10.1016/s0004-3702\(97\)00063-5](https://doi.org/10.1016/s0004-3702(97)00063-5)
- [42] Raymond Bond, Ansgar Koene, Alan Dix, Jennifer Boger, Maurice D. Mulvenna, Mykola Galushka, Bethany Waterhouse Bradley, Fiona Browne, Hui Wang, and Alexander Wong. 2019. Democratisation of Usable Machine Learning in Computer Vision. *ArXiv* (2019). arXiv:cs.CV/1902.06804v1
- [43] Abdelhamid Bouchachia, Bogdan Gabrys, and Zoheir Sahel. 2007. Overview of Some Incremental Learning Algorithms. In *2007 IEEE International Fuzzy Systems Conference*. IEEE. <https://doi.org/10.1109/fuzzy.2007.4295640>
- [44] Pvael Brazdil. 1998. Data transformation and model selection by experimentation and meta-learning. In *Proceedings of the ECML-98 Workshop on Upgrading Learning to Meta-Level: Model Selection and Data Transformation*. 11–17.
- [45] Pavel Brazdil and Christophe Giraud-Carrier. 2017. Metalearning and Algorithm Selection: progress, state of the art and introduction to the 2018 Special Issue. *Machine Learning* 107, 1 (dec 2017), 1–14. <https://doi.org/10.1007/s10994-017-5692-y>
- [46] Pavel B. Brazdil, Carlos Soares, and Joaquim Pinto da Costa. 2003. Ranking Learning Algorithms: Using IBL and Meta-Learning on Accuracy and Time Results. *Machine Learning* 50, 3 (2003), 251–277. <https://doi.org/10.1023/a:1021713901879>
- [47] Leo Breiman. 2001. Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author). *Statist. Sci.* 16, 3 (aug 2001), 199–231. <https://doi.org/10.1214/ss/1009213726>
- [48] Eric Brochu, Vlad M. Cora, and Nando de Freitas. 2010. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *ArXiv* (December 2010). arXiv:cs.LG/1012.2599v1
- [49] Ann L Brown and Mary Jo Kane. 1988. Preschool children can learn to transfer: Learning to learn and learning from example. *Cognitive Psychology* 20, 4 (oct 1988), 493–523. [https://doi.org/10.1016/0010-0285\(88\)90014-x](https://doi.org/10.1016/0010-0285(88)90014-x)
- [50] Marcin Budka and Bogdan Gabrys. 2010. Correntropy-based density-preserving data sampling as an alternative to standard cross-validation. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*. IEEE. <https://doi.org/10.1109/ijcnn.2010.5596717>
- [51] M. Budka and B. Gabrys. 2013. Density-Preserving Sampling: Robust and Efficient Alternative to Cross-Validation for Error Estimation. *IEEE Transactions on Neural Networks and Learning Systems* 24, 1 (jan 2013), 22–34. <https://doi.org/10.1109/tnnls.2012.2222925>
- [52] Marcin Budka, Bogdan Gabrys, and Katarzyna Musial. 2011. On Accuracy of PDF Divergence Estimators and Their Applicability to Representative Data Sampling. *Entropy* 13, 7 (jul 2011), 1229–1266. <https://doi.org/10.3390/e13071229>
- [53] Miguel Cachada, Salisu Abdulrahman, and Pavel Brazdil. 2017. Combining Feature and Algorithm Hyperparameter Selection using some Metalearning Methods. In *AutoML@PKDD/ECML*.
- [54] Silviu Cacoveanu, Camelia Vidrighin, and Rodica Potolea. 2009. Evolutional meta-learning framework for automatic classifier selection. In *2009 IEEE 5th International Conference on Intelligent Computer Communication and Processing*. IEEE, 27–30. <https://doi.org/10.1109/iccp.2009.5284790>
- [55] Michel Camilleri, Filippo Neri, and Michalis Papoutsidakis. 2014. An algorithmic approach to parameter selection in machine learning using meta-optimization techniques. *WSEAS Transactions on systems* 13, 2014 (2014), 202–213.
- [56] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. 2004. Ensemble selection from libraries of models. In *Twenty-first international conference on Machine learning - ICML '04*. ACM Press. <https://doi.org/10.1145/1015330.1015432>

- [57] Bilge Celik and Joaquin Vanschoren. 2020. Adaptation Strategies for Automated Machine Learning on Evolving Data. *ArXiv* (2020). arXiv:cs.LG/2006.06480v1
- [58] Philip K. Chan and Salvatore J. Stolfo. 1993. Experiments on multistrategy learning by meta-learning. In *Proceedings of the second international conference on Information and knowledge management - CIKM '93*. ACM Press. <https://doi.org/10.1145/170088.170160>
- [59] Simon Chan, Philip Treleaven, and Licia Capra. 2013. Continuous hyperparameter optimization for large-scale recommender systems. In *2013 IEEE International Conference on Big Data*. IEEE. <https://doi.org/10.1109/bigdata.2013.6691595>
- [60] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *Comput. Surveys* 41, 3 (jul 2009), 1–58. <https://doi.org/10.1145/1541880.1541882>
- [61] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rudiger Wirth. 2000. *CRISP-DM 1.0: Step-by-step data mining guide*. SPSS.
- [62] A. Charnes, W.W. Cooper, and E. Rhodes. 1978. Measuring the efficiency of decision making units. *European Journal of Operational Research* 2, 6 (nov 1978), 429–444. [https://doi.org/10.1016/0377-2217\(78\)90138-8](https://doi.org/10.1016/0377-2217(78)90138-8)
- [63] Lena Chekina, Lior Rokach, and Bracha Shapira. 2011. Meta-learning for Selecting a Multi-label Classification Algorithm. In *2011 IEEE 11th International Conference on Data Mining Workshops*. IEEE. <https://doi.org/10.1109/icdmw.2011.118>
- [64] Boyuan Chen, Harvey Wu, Warren Mo, Ishanu Chattopadhyay, and Hod Lipson. 2018. Autostacker: A compositional evolutionary learning system. In *Proceedings of the Genetic and Evolutionary Computation Conference - GECCO '18*. ACM Press. <https://doi.org/10.1145/3205455.3205586>
- [65] Justin Cheng and Michael S. Bernstein. 2015. Flock: Hybrid Crowd-Machine Learning Classifiers. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing - CSCW '15*. ACM Press. <https://doi.org/10.1145/2675133.2675214>
- [66] Weiwei Cheng, Gjergji Kasneci, Thore Graepel, David Stern, and Ralf Herbrich. 2011. Automated feature generation from structured knowledge. In *Proceedings of the 20th ACM international conference on Information and knowledge management - CIKM '11*. ACM Press. <https://doi.org/10.1145/2063576.2063779>
- [67] Krzysztof J. Cios and Lukasz A. Kurgan. 2005. Trends in Data Mining and Knowledge Discovery. In *Advanced Information and Knowledge Processing*. Springer London, 1–26. https://doi.org/10.1007/1-84628-183-0_1
- [68] Marc Claesen and Bart De Moor. 2015. Hyperparameter search in machine learning. *Proc. of the 11th Metaheuristics International Conference* abs/1502.02127, 1–5. <https://lirias.kuleuven.be/retrieve/314688> <https://doi.org/10.1016/j.eswa.2015.10.021>
- [69] Robert T. Clemen. 1989. Combining forecasts: A review and annotated bibliography. *International Journal of Forecasting* 5, 4 (jan 1989), 559–583. [https://doi.org/10.1016/0169-2070\(89\)90012-5](https://doi.org/10.1016/0169-2070(89)90012-5)
- [70] Dermot Cochran, Martin Roe, CV Goudar, Breda Martyn, Mainak Das, Pratush Uppuluri, Sean Lennon, Basudev Panda, Sarah King, Liam Dolan, et al. 2018. A methodology for automated feature engineering in training of shallow neural networks for prediction of multi-linear growth failures (“Nova Algorithm”). *Technical Disclosure Commons* (2018).
- [71] ConversionLogic. 2016. *Whitepaper: Multi-Stage Ensemble and Feature Engineering*. Technical Report.
- [72] Daniel Crankshaw, Gur-Eyal Sela, Corey Zumar, Xiangxi Mo, Joseph E. Gonzalez, Ion Stoica, and Alexey Tumanov. 2018. InferLine: ML Inference Pipeline Composition Framework. *ArXiv* (December 2018). arXiv:cs.DC/1812.01776v1
- [73] Can Cui, Mengqi Hu, Jeffery D. Weir, and Teresa Wu. 2016. A recommendation system for meta-modeling: A meta-learning based approach. *Expert Systems with Applications* 46 (mar 2016), 33–44. <https://doi.org/10.1016/j.eswa.2015.10.021>
- [74] Henggang Cui, Gregory R. Ganger, and Phillip B. Gibbons. 2018. MLtuner: System Support for Automatic Machine Learning Tuning. *ArXiv* (March 2018). arXiv:cs.LG/1803.07445v1
- [75] Aron Culotta, Trausti Kristjansson, Andrew McCallum, and Paul Viola. 2006. Corrective feedback and persistent learning for information extraction. *Artificial Intelligence* 170, 14–15 (oct 2006), 1101–1122. <https://doi.org/10.1016/j.artint.2006.08.001>
- [76] Alfonso de la Vega, Diego García-Saiz, Marta Zorrilla, and Pablo Sánchez. 2019. How Far are we from Data Mining Democratisation? A Systematic Review. *ArXiv* (2019). arXiv:cs.DB/1903.08431v1
- [77] Alex G. C. de Sá, Walter José G. S. Pinto, Luiz Otavio V. B. Oliveira, and Gisele L. Pappa. 2017. RECIPE: A Grammar-Based Framework for Automatically Evolving Classification Pipelines. In *Lecture Notes in Computer Science*. Springer International Publishing, 246–261. https://doi.org/10.1007/978-3-319-55696-3_16
- [78] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (apr 2002), 182–197. <https://doi.org/10.1109/4235.996017>
- [79] Ian Dewancker, Michael McCourt, Scott Clark, Patrick Hayes, Alexandra Johnson, and George Ke. 2016. A strategy for ranking optimization methods using multiple criteria. In *Workshop on Automatic Machine Learning*. 11–20.

- [80] Nilanjan Dey, Aboul Ella Hassanien, Chintan Bhatt, Amira S. Ashour, and Suresh Chandra Satapathy (Eds.). 2018. *Internet of Things and Big Data Analytics Toward Next-Generation Intelligence*. Springer International Publishing. <https://doi.org/10.1007/978-3-319-60435-0>
- [81] A.I. Diveev, S.V. Konstantinov, and E.A. Sofronova. 2018. A Comparison of Evolutionary Algorithms and Gradient-based Methods for the Optimal Control Problem. In *2018 5th International Conference on Control, Decision and Information Technologies (CoDIT)*. IEEE. <https://doi.org/10.1109/codit.2018.8394805>
- [82] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. 2015. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.
- [83] Xuanyi Dong, Lu Liu, Katarzyna Musial, and Bogdan Gabrys. 2020. NATS-Bench: Benchmarking NAS Algorithms for Architecture Topology and Size. *ArXiv (2020)*. arXiv:cs.LG/2009.00437v4
- [84] Xuanyi Dong, Mingxing Tan, Adams Wei Yu, Daiyi Peng, Bogdan Gabrys, and Quoc V. Le. 2020. AutoHAS: Differentiable Hyper-parameter and Architecture Search. *ArXiv (2020)*. arXiv:cs.CV/2006.03656v1
- [85] Ofer Dor and Yoram Reich. 2012. Strengthening learning algorithms by feature discovery. *Information Sciences* 189 (apr 2012), 176–190. <https://doi.org/10.1016/j.ins.2011.11.039>
- [86] Iddo Drori, Yamuna Krishnamurthy, Remi Rampin, Raoni Lourenço, J One, Kyunghyun Cho, Claudio Silva, and Juliana Freire. 2018. AlphaD3M: Machine learning pipeline synthesis. In *AutoML Workshop at ICML*.
- [87] Jaimie Drozdal, Justin Weisz, Dakuo Wang, Gaurav Dass, Bingsheng Yao, Changruo Zhao, Michael Muller, Lin Ju, and Hui Su. 2020. Trust in AutoML: exploring information needs for establishing trust in automated machine learning systems. In *Proceedings of the 25th International Conference on Intelligent User Interfaces*. ACM. <https://doi.org/10.1145/3377325.3377501> arXiv:cs.LG/2001.06509v1
- [88] Sébastien Dubois. 2015. *Deep mining: Copula-based hyper-parameter optimization for machine learning pipelines*. Master's thesis. École polytechnique - Massachusetts Institute of Technology.
- [89] Celestine Dunner, Thomas Parnell, Kubilay Atasu, Manolis Sifalakis, and Haralampos Pozidis. 2017. Understanding and optimizing the performance of distributed machine learning applications on apache spark. In *2017 IEEE International Conference on Big Data (Big Data)*. IEEE. <https://doi.org/10.1109/bigdata.2017.8257942>
- [90] Salvador Dura-Bernal, Benjamin A Suter, Pdraig Gleeson, Matteo Cantarelli, Adrian Quintana, Facundo Rodriguez, David J Kedziora, George L Chadderdon, Cliff C Kerr, Samuel A Neymotin, Robert A McDougal, Michael Hines, Gordon MG Shepherd, and William W Lytton. 2019. NetPyNE, a tool for data-driven multiscale modeling of brain circuits. *eLife* 8 (apr 2019). <https://doi.org/10.7554/eLife.44494>
- [91] Mark Eastwood and Bogdan Gabrys. 2007. The Dynamics of Negative Correlation Learning. *The Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology* 49, 2 (jul 2007), 251–263. <https://doi.org/10.1007/s11265-007-0074-5>
- [92] Mark Eastwood and Bogdan Gabrys. 2011. Model level combination of tree ensemble hyperboxes via GFMM. In *2011 Eighth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*. IEEE. <https://doi.org/10.1109/fskd.2011.6019563>
- [93] Mark Eastwood and Bogdan Gabrys. 2012. Generalised bottom-up pruning: A model level combination of decision trees. *Expert Systems with Applications* 39, 10 (aug 2012), 9150–9158. <https://doi.org/10.1016/j.eswa.2012.02.061>
- [94] Valeria Efimova, Andrey Filchenkov, and Anatoly Shalyto. 2019. Reinforcement-Based Simultaneous Algorithm and Its Hyperparameters Selection. In *Communications in Computer and Information Science*. Springer International Publishing, 15–27. https://doi.org/10.1007/978-3-030-35400-8_2
- [95] Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger Hoos, and Kevin Leyton-Brown. 2013. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, Vol. 10. 3.
- [96] 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*.
- [97] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural Architecture Search: A Survey. *Journal of Machine Learning Research* 20, 55 (2019), 1–21. <http://jmlr.org/papers/v20/18-598.html>
- [98] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. 2020. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. *ArXiv (2020)*. arXiv:stat.ML/2003.06505v1
- [99] Hugo Jair Escalante, Manuel Montes, and Luis Enrique Sucar. 2009. Particle Swarm Model Selection. *J. Mach. Learn. Res.* 10 (June 2009), 405–440.
- [100] Rasool Fakoor, Jonas Mueller, Nick Erickson, Pratik Chaudhari, and Alexander J. Smola. 2020. Fast, Accurate, and Simple Models for Tabular Data via Augmented Distillation. *Advances in Neural Information Processing Systems* 33 (2020). arXiv:cs.LG/2006.14284v1
- [101] Stefan Falkner, Aaron Klein, and Frank Hutter. 2017. Combining hyperband and bayesian optimization. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS), Bayesian Optimization Workshop*.

- [102] Meng Fang, Yuan Li, and Trevor Cohn. 2017. Learning how to Active Learn: A Deep Reinforcement Learning Approach. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. <https://doi.org/10.18653/v1/d17-1063>
- [103] J. Doyne Farmer, Norman H Packard, and Alan S Perelson. 1986. The immune system, adaptation, and machine learning. *Physica D: Nonlinear Phenomena* 22, 1-3 (oct 1986), 187–204. [https://doi.org/10.1016/0167-2789\(86\)90240-x](https://doi.org/10.1016/0167-2789(86)90240-x)
- [104] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. 1996. The KDD process for extracting useful knowledge from volumes of data. *Commun. ACM* 39, 11 (nov 1996), 27–34. <https://doi.org/10.1145/240455.240464>
- [105] Susana Fernández, Tomás de la Rosa, Fernando Fernández, Rubén Suárez, Javier Ortiz, Daniel Borrajo, and David Manzano. 2013. Using automated planning for improving data mining processes. *The Knowledge Engineering Review* 28, 2 (feb 2013), 157–173. <https://doi.org/10.1017/s0269888912000409>
- [106] Matthias Feurer, Aaron Klein, Katharina Eggenberger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and Robust Automated Machine Learning. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 2962–2970. <http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf>
- [107] Matthias Feurer, Aaron Klein, Katharina Eggenberger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. 2019. Auto-sklearn: Efficient and Robust Automated Machine Learning. In *Automated Machine Learning*. Springer International Publishing, 113–134. https://doi.org/10.1007/978-3-030-05318-5_6
- [108] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. 2015. Initializing bayesian hyperparameter optimization via meta-learning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.
- [109] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70 (ICML'17)*. JMLR.org, 1126–1135.
- [110] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. 2017. One-Shot Visual Imitation Learning via Meta-Learning. In *Proceedings of the 1st Annual Conference on Robot Learning (Proceedings of Machine Learning Research)*, Sergey Levine, Vincent Vanhoucke, and Ken Goldberg (Eds.), Vol. 78. PMLR, 357–368. <http://proceedings.mlr.press/v78/finn17a.html>
- [111] Pablo Fonseca, Julio Mendoza, Jacques Wainer, Jose Ferrer, Joseph Pinto, Jorge Guerrero, and Benjamin Castaneda. 2015. Automatic breast density classification using a convolutional neural network architecture search procedure. In *Medical Imaging 2015: Computer-Aided Diagnosis*, Lubomir M. Hadjiiski and Georgia D. Tourassi (Eds.). SPIE. <https://doi.org/10.1117/12.2081576>
- [112] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, Stephen Heil, Prerak Patel, Adam Sapek, Gabriel Weisz, Lisa Woods, Sitaram Lanka, Steven K. Reinhardt, Adrian M. Caulfield, Eric S. Chung, and Doug Burger. 2018. A Configurable Cloud-Scale DNN Processor for Real-Time AI. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. <https://doi.org/10.1109/isca.2018.00012>
- [113] Yonggan Fu, Wuyang Chen, Haotao Wang, Haoran Li, Yingyan Lin, and Zhangyang Wang. 2020. AutoGAN-Distiller: Searching to Compress Generative Adversarial Networks. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Hal Daumé III and Aarti Singh (Eds.), Vol. 119. PMLR, Virtual, 3292–3303. <http://proceedings.mlr.press/v119/fu20b.html>
- [114] Ben D. Fulcher and Nick S. Jones. 2014. Highly Comparative Feature-Based Time-Series Classification. *IEEE Transactions on Knowledge and Data Engineering* 26, 12 (dec 2014), 3026–3037. <https://doi.org/10.1109/tkde.2014.2316504>
- [115] Johannes Fürnkranz and Johann Petrak. 2001. An evaluation of landmarking variants. In *Working Notes of the ECML/PKDD 2000 Workshop on Integrating Aspects of Data Mining, Decision Support and Meta-Learning*. 57–68.
- [116] B. Gabrys. 2002. Combining neuro-fuzzy classifiers for improved generalisation and reliability. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*. IEEE. <https://doi.org/10.1109/ijcnn.2002.1007519>
- [117] Bogdan Gabrys. 2004. Learning hybrid neuro-fuzzy classifier models from data: to combine or not to combine? *Fuzzy Sets and Systems* 147, 1 (oct 2004), 39–56. <https://doi.org/10.1016/j.fss.2003.11.010>
- [118] Bogdan Gabrys and Andrzej Bargiela. 1999. Neural Networks Based Decision Support in Presence of Uncertainties. *Journal of Water Resources Planning and Management* 125, 5 (sep 1999), 272–280. [https://doi.org/10.1061/\(asce\)0733-9496\(1999\)125:5\(272\)](https://doi.org/10.1061/(asce)0733-9496(1999)125:5(272))
- [119] Bogdan Gabrys, Kauko Leiviskä, and Jens Strackeljan (Eds.). 2005. *Do Smart Adaptive Systems Exist?* Springer Berlin Heidelberg. <https://doi.org/10.1007/3-540-32374-0>
- [120] Bogdan Gabrys and Dymitr Ruta. 2006. Genetic algorithms in classifier fusion. *Applied Soft Computing* 6, 4 (aug 2006), 337–347. <https://doi.org/10.1016/j.asoc.2005.11.001>

- [121] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. 2004. Learning with Drift Detection. In *Advances in Artificial Intelligence – SBIA 2004*. Springer Berlin Heidelberg, 286–295. https://doi.org/10.1007/978-3-540-28645-5_29
- [122] João Gama, Indrè Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *Comput. Surveys* 46, 4 (mar 2014), 1–37. <https://doi.org/10.1145/2523813>
- [123] Yaroslav Ganin and Victor Lempitsky. 2015. Unsupervised Domain Adaptation by Backpropagation. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Francis Bach and David Blei (Eds.), Vol. 37. PMLR, Lille, France, 1180–1189. <http://proceedings.mlr.press/v37/ganin15.html>
- [124] Unai Garcíarena, Alexander Mendiburu, and Roberto Santana. 2018. Towards a more efficient representation of imputation operators in TPOT. *ArXiv* (January 2018). arXiv:cs.LG/1801.04407v1
- [125] Unai Garcíarena, Roberto Santana, and Alexander Mendiburu. 2017. Evolving imputation strategies for missing data in classification problems with TPOT. *ArXiv* (June 2017). arXiv:cs.LG/1706.01120v2
- [126] Unai Garcíarena, Roberto Santana, and Alexander Mendiburu. 2018. Analysis of the Complexity of the Automatic Pipeline Generation Problem. In *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. <https://doi.org/10.1109/cec.2018.8477662>
- [127] Mike Gashler, Christophe Giraud-Carrier, and Tony Martinez. 2008. Decision Tree Ensemble: Small Heterogeneous Is Better Than Large Homogeneous. In *2008 Seventh International Conference on Machine Learning and Applications*. IEEE. <https://doi.org/10.1109/icmla.2008.154>
- [128] David Gelernter. 1991. *Mirror worlds: or the day software puts the universe in a shoebox—how it will happen and what it will mean*. Oxford University Press, New York.
- [129] Stuart Geman, Elie Bienenstock, and René Doursat. 1992. Neural Networks and the Bias/Variance Dilemma. *Neural Computation* 4, 1 (jan 1992), 1–58. <https://doi.org/10.1162/neco.1992.4.1.1>
- [130] T van Gemert. 2017. *On the influence of dataset characteristics on classifier performance*. B.S. thesis. Utrecht University.
- [131] Ian Gemp, Georgios Theodorou, and Mohammad Ghavamzadeh. 2017. Automated data cleansing through meta-learning. In *Twenty-Ninth IAAI Conference*.
- [132] Oguzhan Gencoglu, Mark van Gils, Esin Guldogan, Chamin Morikawa, Mehmet Süzen, Mathias Gruber, Jussi Leinonen, and Heikki Huttunen. 2019. HARK Side of Deep Learning – From Grad Student Descent to Automated Machine Learning. *ArXiv* (2019). arXiv:cs.LG/1904.07633v1
- [133] Efstathios D. Gennatas, Jerome H. Friedman, Lyle H. Ungar, Romain Pirracchio, Eric Eaton, Lara G. Reichmann, Yannet Interian, José Marcio Luna, Charles B. Simone, Andrew Auerbach, Elier Delgado, Mark J. van der Laan, Timothy D. Solberg, and Gilmer Valdes. 2020. Expert-augmented machine learning. *Proceedings of the National Academy of Sciences* 117, 9 (feb 2020), 4571–4577. <https://doi.org/10.1073/pnas.1906831117>
- [134] Mike Gerdes, Diego Galar, and Dieter Scholz. 2016. Automated parameter optimization for feature extraction for condition monitoring. In *14th IMEKO TC10 Workshop on Technical Diagnostics 2016: New Perspectives in Measurements, Tools and Techniques for Systems Reliability, Maintainability and Safety, Milan, Italy, 27-28 June 2016*. 452–457.
- [135] Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. 2019. An Open Source AutoML Benchmark. *ArXiv* (2019). arXiv:cs.LG/1907.00909v1
- [136] Pieter Gijsbers and Joaquin Vanschoren. 2019. GAMA: Genetic Automated Machine learning Assistant. *Journal of Open Source Software* 4, 33 (jan 2019), 1132. <https://doi.org/10.21105/joss.01132>
- [137] Yolanda Gil, James Honaker, Shikhar Gupta, Yibo Ma, Vito D’Orazio, Daniel Garijo, Shruti Gadewar, Qifan Yang, and Neda Jahanshad. 2019. Towards human-guided machine learning. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*. ACM. <https://doi.org/10.1145/3301275.3302324>
- [138] Assaf Glazer and Shaul Markovitch. 2013. *Resource-Bounded Selection of Learning Algorithms*. Technical Report.
- [139] Taciana A.F. Gomes, Ricardo B.C. Prudêncio, Carlos Soares, André L.D. Rossi, and André Carvalho. 2012. Combining meta-learning and search techniques to select parameters for support vector machines. *Neurocomputing* 75, 1 (jan 2012), 3–13. <https://doi.org/10.1016/j.neucom.2011.07.005>
- [140] Xinyu Gong, Shiyu Chang, Yifan Jiang, and Zhangyang Wang. 2019. AutoGAN: Neural Architecture Search for Generative Adversarial Networks. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE. <https://doi.org/10.1109/iccv.2019.00332>
- [141] Krzysztof Grańbczewski. 2014. *Meta-Learning in Decision Tree Induction*. Springer International Publishing. <https://doi.org/10.1007/978-3-319-00960-5>
- [142] Krzysztof Grańbczewski and Norbert Jankowski. 2007. Versatile and Efficient Meta-Learning Architecture: Knowledge Representation and Management in Computational Intelligence. In *2007 IEEE Symposium on Computational Intelligence and Data Mining*. IEEE. <https://doi.org/10.1109/cidm.2007.368852>
- [143] Isabelle Guyon, Imad Chaabane, Hugo Jair Escalante, Sergio Escalera, Damir Jajetic, James Robert Lloyd, Núria Macià, Bisakha Ray, Lukasz Romaszko, Michèle Sebag, Alexander Statnikov, Sébastien Treguer, and Evelyne Viegas. 2016. A brief review of the ChaLearn AutoML challenge: any-time any-dataset learning without human intervention. In *Workshop on Automatic Machine Learning*. 21–30.

- [144] Isabelle Guyon, Amir Saffari, Gideon Dror, and Gavin Cawley. 2010. Model selection: Beyond the bayesian/frequentist divide. *Journal of Machine Learning Research* 11, Jan (2010), 61–87.
- [145] Isabelle Guyon, Amir Saffari, Gideon Dror, and Gavin Cawley. 2011. Challenges in Data Representation, Model Selection, and Performance Prediction. In *Hands-On Pattern Recognition Challenges in Machine Learning, Volume 1*. Microtome Publishing USA, Chapter 1.
- [146] H2O.ai. 2017. *H2O AutoML*. <http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html> H2O version 3.30.0.1.
- [147] P. M. Hall, D. Marshall, and R. R. Martin. 1998. Incremental Eigenanalysis for Classification. In *Proceedings of the British Machine Vision Conference 1998*. British Machine Vision Association. <https://doi.org/10.5244/c.12.29>
- [148] Torben Hansing, Mario Michael Krell, and Frank Kirchner. 2016. hyperSPACE: Automated Optimization of Complex Processing Pipelines for pySPACE. In *NIPS Workshop on Bayesian Optimization. NIPS Workshop on Bayesian Optimization (BayesOPT2016), December 5-10, Barcelona, Spain*.
- [149] Marc Hanussek, Matthias Blohm, and Maximilien Kintz. 2020. Can AutoML outperform humans? An evaluation on popular OpenML datasets using AutoML Benchmark. *ArXiv* (2020). arXiv:cs.LG/2009.01564
- [150] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. 2018. PipeDream: Fast and Efficient Pipeline Parallel DNN Training. *ArXiv* (June 2018). arXiv:cs.DC/1806.03377v1
- [151] Chaoyang He, Murali Annavam, and Salman Avestimehr. 2020. FedNAS: Federated Deep Learning via Neural Architecture Search. *ArXiv* (2020). arXiv:cs.LG/2004.08546
- [152] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2014. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. In *Computer Vision – ECCV 2014*. Springer International Publishing, 346–361. https://doi.org/10.1007/978-3-319-10578-9_23
- [153] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. <https://doi.org/10.1109/cvpr.2016.90>
- [154] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems* 212 (jan 2021), 106622. <https://doi.org/10.1016/j.knsys.2020.106622> arXiv:cs.LG/1908.00709
- [155] Jeff T Heaton. 2017. *Automated Feature Engineering for Deep Neural Networks with Genetic Programming*. Ph.D. Dissertation. College of Engineering and Computing, Nova Southeastern University.
- [156] Philipp Hennig and Christian J. Schuler. 2012. Entropy Search for Information-Efficient Global Optimization. *J. Mach. Learn. Res.* 13, null (June 2012), 1809–1837.
- [157] Lars Hertel, Julian Collado, Peter J. Sadowski, and Pierre Baldi. 2018. Sherpa: Hyperparameter Optimization for Machine Learning Models. In *32nd Conference on Neural Information Processing Systems (NIPS 2018)*.
- [158] Tin Kam Ho and M. Basu. 2002. Complexity measures of supervised classification problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24, 3 (mar 2002), 289–300. <https://doi.org/10.1109/34.990132>
- [159] Matthew Hoffman, Bobak Shahriari, and Nando Freitas. 2014. On correlation and budget constraints in model-based bandit optimization with application to automatic machine learning. In *Artificial Intelligence and Statistics*. 365–374.
- [160] Holger H. Hoos, Frank Neumann, and Heike Trautmann. 2017. Automated Algorithm Selection and Configuration (Dagstuhl Seminar 16412). *Dagstuhl Reports* 6, 10 (2017), 33–74. <https://doi.org/10.4230/DagRep.6.10.33>
- [161] Daniel Horn, Tobias Wagner, Dirk Biermann, Claus Weihs, and Bernd Bischl. 2015. Model-Based Multi-objective Optimization: Taxonomy, Multi-Point Proposal, Toolbox and Benchmark. In *Lecture Notes in Computer Science*. Springer International Publishing, 64–78. https://doi.org/10.1007/978-3-319-15934-8_5
- [162] Véronique Hoste, Iris Hendrickx, Walter Daelemans, and Antal van den Bosch. 2002. Parameter optimization for machine-learning of word sense disambiguation. *Natural Language Engineering* 8, 4 (dec 2002), 311–325. <https://doi.org/10.1017/s1351324902003005>
- [163] Hanzhang Hu, John Langford, Rich Caruana, Eric Horvitz, and Debadeepta Dey. 2018. Macro Neural Architecture Search Revisited. In *2nd Workshop on Meta-Learning at NeurIPS*.
- [164] Yuh-Jong Hu and Shu-Wei Huang. 2017. Challenges of automated machine learning on causal impact analytics for policy evaluation. In *2017 2nd International Conference on Telecommunication and Networks (TEL-NET)*. IEEE. <https://doi.org/10.1109/tel-net.2017.8343571>
- [165] Frank Hutter, Thomas Bartz-Beielstein, Holger H. Hoos, Kevin Leyton-Brown, and Kevin P. Murphy. 2010. Sequential Model-Based Parameter Optimization: an Experimental Investigation of Automated and Interactive Approaches. In *Experimental Methods for the Analysis of Optimization Algorithms*. Springer Berlin Heidelberg, 363–414. https://doi.org/10.1007/978-3-642-02538-9_15
- [166] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. 2014. An Efficient Approach for Assessing Hyperparameter Importance. In *Proceedings of the 31st International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Eric P. Xing and Tony Jebara (Eds.), Vol. 32. PMLR, Beijing, China, 754–762. <http://proceedings.mlr.press/v32/hutter14.html>
- [167] 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 Berlin Heidelberg,

- 507–523. https://doi.org/10.1007/978-3-642-25566-3_40
- [168] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Kevin P. Murphy. 2009. An experimental investigation of model-based parameter optimisation: SPO and beyond. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation - GECCO '09*. ACM Press. <https://doi.org/10.1145/1569901.1569940>
- [169] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stuetzle. 2009. ParamILS: An Automatic Algorithm Configuration Framework. *Journal of Artificial Intelligence Research* 36 (oct 2009), 267–306. <https://doi.org/10.1613/jair.2861>
- [170] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (Eds.). 2019. *Automated Machine Learning: Methods, Systems, Challenges*. Springer International Publishing. <https://doi.org/10.1007/978-3-030-05318-5>
- [171] Frank Hutter, Jörg Lücke, and Lars Schmidt-Thieme. 2015. Beyond Manual Tuning of Hyperparameters. *KI - Künstliche Intelligenz* 29, 4 (jul 2015), 329–337. <https://doi.org/10.1007/s13218-015-0381-0>
- [172] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. 2014. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence* 206 (jan 2014), 79–111. <https://doi.org/10.1016/j.artint.2013.10.003>
- [173] A. Imbrea. 2020. An empirical comparison of automated machine learning techniques for data streams. <http://essay.utwente.nl/80548/>
- [174] Kevin Jamieson and Ameet Talwalkar. 2016. Non-stochastic Best Arm Identification and Hyperparameter Optimization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Arthur Gretton and Christian C. Robert (Eds.), Vol. 51. PMLR, Cadiz, Spain, 240–248. <http://proceedings.mlr.press/v51/jamieson16.html>
- [175] Norbert Jankowski, Włodzisław Duch, and Krzysztof Grafczewski. 2011. *Meta-learning in computational intelligence*. Vol. 358. Springer.
- [176] Norbert Jankowski and Krzysztof Grafczewski. 2008. Building meta-learning algorithms basing on search controlled by machine complexity. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. IEEE. <https://doi.org/10.1109/ijcnn.2008.4634313>
- [177] Norbert Jankowski and Krzysztof Grafczewski. 2011. Universal Meta-Learning Architecture and Algorithms. In *Studies in Computational Intelligence*. Springer Berlin Heidelberg, 1–76. https://doi.org/10.1007/978-3-642-20980-2_1
- [178] Sergio Jiménez, Tomás De La Rosa, Susana Fernández, Fernando Fernández, and Daniel Borrajo. 2012. A review of machine learning for automated planning. *The Knowledge Engineering Review* 27, 4 (nov 2012), 433–467. <https://doi.org/10.1017/s026988891200001x>
- [179] Haifeng Jin, Qingquan Song, and Xia Hu. 2019. Auto-Keras: An Efficient Neural Architecture Search System. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM. <https://doi.org/10.1145/3292500.3330648> arXiv:cs.LG/1806.10282v3
- [180] Donald R. Jones, Matthias Schonlau, and William J. Welch. 1998. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization* 13, 4 (1998), 455–492. <https://doi.org/10.1023/a:1008306431147>
- [181] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Soutter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ACM. <https://doi.org/10.1145/3079856.3080246>
- [182] Joseph B Kadane and Nicole A Lazar. 2004. Methods and Criteria for Model Selection. *J. Amer. Statist. Assoc.* 99, 465 (mar 2004), 279–290. <https://doi.org/10.1198/016214504000000269>
- [183] Petr Kadlec and Bogdan Gabrys. 2009. Architecture for development of adaptive on-line prediction models. *Memetic Computing* 1, 4 (sep 2009), 241–269. <https://doi.org/10.1007/s12293-009-0017-8>
- [184] Petr Kadlec and Bogdan Gabrys. 2009. Evolving on-line prediction model dealing with industrial data sets. In *2009 IEEE Workshop on Evolving and Self-Developing Intelligent Systems*. IEEE. <https://doi.org/10.1109/esdis.2009.4938995>
- [185] Petr Kadlec and Bogdan Gabrys. 2009. Soft Sensor Based on Adaptive Local Learning. In *Advances in Neuro-Information Processing*. Springer Berlin Heidelberg, 1172–1179. https://doi.org/10.1007/978-3-642-02490-0_142
- [186] Petr Kadlec and Bogdan Gabrys. 2009. Soft sensors: where are we and what are the current and future challenges? *IFAC Proceedings Volumes* 42, 19 (2009), 572–577. <https://doi.org/10.3182/20090921-3-tr-3005.00098>
- [187] Petr Kadlec and Bogdan Gabrys. 2010. Adaptive on-line prediction soft sensing without historical data. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*. IEEE. <https://doi.org/10.1109/ijcnn.2010.5596965>

- [188] Petr Kadlec and Bogdan Gabrys. 2010. Local learning-based adaptive soft sensor for catalyst activation prediction. *AIChE Journal* 57, 5 (jun 2010), 1288–1301. <https://doi.org/10.1002/aic.12346>
- [189] Petr Kadlec, Ratko Grbić, and Bogdan Gabrys. 2011. Review of adaptation mechanisms for data-driven soft sensors. *Computers & Chemical Engineering* 35, 1 (jan 2011), 1–24. <https://doi.org/10.1016/j.compchemeng.2010.07.034>
- [190] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D’Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrede Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. 2019. Advances and Open Problems in Federated Learning. *ArXiv* (2019). arXiv:cs.LG/1912.04977
- [191] Vasiliki Kalavri. 2016. *Performance optimization techniques and tools for distributed graph processing*. Ph.D. Dissertation. KTH Royal Institute of Technology.
- [192] James Max Kanter and Kalyan Veeramachaneni. 2015. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE. <https://doi.org/10.1109/dsaa.2015.7344858>
- [193] N. Kasabov. 2001. Evolving fuzzy neural networks for supervised/unsupervised online knowledge-based learning. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)* 31, 6 (2001), 902–918. <https://doi.org/10.1109/3477.969494>
- [194] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. 2016. ExploreKit: Automatic Feature Generation and Selection. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE. <https://doi.org/10.1109/icdm.2016.0123>
- [195] Thanh Tung Khuat, Fang Chen, and Bogdan Gabrys. 2020. An improved online learning algorithm for general fuzzy min-max neural network. *ArXiv* (2020). arXiv:cs.LG/2001.02391v1
- [196] Udayan Khurana, Fatemeh Nargesian, Horst Samulowitz, Elias Khalil, and Deepak Turaga. 2016. Automating feature engineering. *Transformation* 10, 10 (2016), 10.
- [197] Udayan Khurana, Horst Samulowitz, and Deepak Turaga. 2017. Feature Engineering for Predictive Modeling using Reinforcement Learning. *ArXiv* (September 2017). arXiv:cs.AI/1709.07150v1
- [198] Udayan Khurana, Deepak Turaga, Horst Samulowitz, and Srinivasan Parthasarathy. 2016. Cognito: Automated Feature Engineering for Supervised Learning. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*. IEEE. <https://doi.org/10.1109/icdmw.2016.0190>
- [199] J-U Kietz, F Serban, A Bernstein, and S Fischer. 2009. Towards cooperative planning of data mining workflows. In *Proc of the ECML/PKDD09 Workshop on Third Generation Data Mining: Towards Service-oriented Knowledge Discovery (SoKD-09)*. <https://doi.org/10.5167/uzh-25868>
- [200] Jörg-Uwe Kietz, Floarea Serban, Abraham Bernstein, and Simon Fischer. 2012. Designing KDD-Workflows via HTN-Planning for Intelligent Discovery Assistance. In *Planning to Learn 2012, Workshop at ECAI 2012*, Joaquin Vanschoren, Jörg-Uwe Kietz, and Pavel Brazdil (Eds.). CEUR Workshop Proceedings. <https://doi.org/10.5167/uzh-67514>
- [201] Jungtaek Kim, Jongheon Jeong, and Seungjin Choi. 2016. AutoML Challenge: AutoML Framework Using Random Space Partitioning Optimizer. In *Workshop on Automated Machine Learning (AutoML)*, ICML. 1–4.
- [202] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. 2017. Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Aarti Singh and Jerry Zhu (Eds.), Vol. 54. PMLR, Fort Lauderdale, FL, USA, 528–536. <http://proceedings.mlr.press/v54/klein17a.html>
- [203] Ron Kohavi and George H. John. 1997. Wrappers for feature subset selection. *Artificial Intelligence* 97, 1-2 (dec 1997), 273–324. [https://doi.org/10.1016/s0004-3702\(97\)00043-x](https://doi.org/10.1016/s0004-3702(97)00043-x)
- [204] J Zico Kolter and Marcus A Maloof. 2007. Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research* 8, Dec (2007), 2755–2790.
- [205] Brent Komer, James Bergstra, and Chris Eliasmith. 2014. Hyperopt-Sklearn: Automatic Hyperparameter Configuration for Scikit-Learn. In *Proceedings of the 13th Python in Science Conference*. SciPy. <https://doi.org/10.25080/majora-14bd3278-006>
- [206] Brent Komer, James Bergstra, and Chris Eliasmith. 2019. *Hyperopt-Sklearn*. Springer International Publishing, 97–111. https://doi.org/10.1007/978-3-030-05318-5_5
- [207] Pavel Kordík, Jan Černý, and Tomáš Frýda. 2017. Discovering predictive ensembles for transfer learning and meta-learning. *Machine Learning* 107, 1 (dec 2017), 177–207. <https://doi.org/10.1007/s10994-017-5682-0>
- [208] Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. 2019. Auto-WEKA: Automatic Model Selection and Hyperparameter Optimization in WEKA. In *Automated Machine Learning*. Springer International

- Publishing, 81–95. https://doi.org/10.1007/978-3-030-05318-5_4
- [209] J.R. Koza. 1990. Genetically breeding populations of computer programs to solve problems in artificial intelligence. In [1990] *Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence*. IEEE Comput. Soc. Press. <https://doi.org/10.1109/tai.1990.130444>
- [210] Tim Kraska. 2018. Northstar: an interactive data science system. *Proceedings of the VLDB Endowment* 11, 12 (aug 2018), 2150–2164. <https://doi.org/10.14778/3229863.3240493>
- [211] Tim Kraska, Ameet Talwalkar, John C Duchi, Rean Griffith, Michael J Franklin, and Michael I Jordan. 2013. MLbase: A Distributed Machine-learning System.. In *CIDR*, Vol. 1. 2–1.
- [212] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems* 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105. <https://doi.org/10.1145/3065386>
- [213] Lukasz A. Kurgan and Petr Musilek. 2006. A survey of Knowledge Discovery and Data Mining process models. *The Knowledge Engineering Review* 21, 1 (mar 2006), 1–24. <https://doi.org/10.1017/s0269888906000737>
- [214] H. J. Kushner. 1964. A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise. *Journal of Basic Engineering* 86, 1 (mar 1964), 97–106. <https://doi.org/10.1115/1.3653121>
- [215] Alexandre Lacoste, Hugo Larochelle, Mario Marchand, and François Laviolette. 2014. Sequential Model-Based Ensemble Optimization. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence (UAI'14)*. AUAI Press, Arlington, Virginia, USA, 440–448.
- [216] Hoang Thanh Lam, Tran Ngoc Minh, Mathieu Sinn, Beat Buesser, and Martin Wistuba. 2018. Neural Feature Learning From Relational Database. *ArXiv* (January 2018). arXiv:cs.AI/1801.05372v4
- [217] Hoang Thanh Lam, Johann-Michael Thiebaut, Mathieu Sinn, Bei Chen, Tiep Mai, and Oznur Alkan. 2017. One button machine for automating feature engineering in relational databases. *ArXiv* (June 2017). arXiv:cs.DB/1706.00327v1
- [218] Hugh Leather, Edwin Bonilla, and Michael O’Boyle. 2009. Automatic Feature Generation for Machine Learning Based Optimizing Compilation. In *2009 International Symposium on Code Generation and Optimization*. IEEE. <https://doi.org/10.1109/cgo.2009.21>
- [219] Doris Jung-Lin Lee, Stephen Macke, Doris Xin, Angela Lee, Silu Huang, and Aditya Parameswaran. 2019. A Human-in-the-loop Perspective on AutoML: Milestones and the Road Ahead. *Data Engineering* 58 (2019).
- [220] Rui Leite and Pavel Brazdil. 2005. Predicting relative performance of classifiers from samples. In *Proceedings of the 22nd international conference on Machine learning - ICML '05*. ACM Press. <https://doi.org/10.1145/1102351.1102414>
- [221] Rui Leite and Pavel Brazdil. 2010. Active Testing Strategy to Predict the Best Classification Algorithm via Sampling and Metalearning. In *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence*. IOS Press, NLD, 309–314.
- [222] Rui Leite, Pavel Brazdil, and Joaquin Vanschoren. 2012. Selecting Classification Algorithms with Active Testing. In *Machine Learning and Data Mining in Pattern Recognition*. Springer Berlin Heidelberg, 117–131. https://doi.org/10.1007/978-3-642-31537-4_10
- [223] Christiane Lemke, Marcin Budka, and Bogdan Gabrys. 2015. Metalearning: a survey of trends and technologies. *Artificial Intelligence Review* 44, 1 (jul 2015), 117–130. <https://doi.org/10.1007/s10462-013-9406-y>
- [224] Christiane Lemke and Bogdan Gabrys. 2008. Do We Need Experts for Time Series Forecasting?. In *16th European Symposium on ArtiProceedings of the ficial Neural Networks (ESANN'2008)*. d-side, 253–258. <http://eprints.bournemouth.ac.uk/8516/>
- [225] Christiane Lemke and Bogdan Gabrys. 2010. Meta-learning for time series forecasting and forecast combination. *Neurocomputing* 73, 10–12 (jun 2010), 2006–2016. <https://doi.org/10.1016/j.neucom.2009.09.020>
- [226] Christiane Lemke and Bogdan Gabrys. 2010. Meta-learning for time series forecasting in the NN GC1 competition. In *International Conference on Fuzzy Systems*. IEEE. <https://doi.org/10.1109/fuzzy.2010.5584001>
- [227] Christiane Lemke, Silvia Riedel, and Bogdan Gabrys. 2012. Evolving forecast combination structures for airline revenue management. *Journal of Revenue and Pricing Management* 12, 3 (aug 2012), 221–234. <https://doi.org/10.1057/rpm.2012.30>
- [228] Julien-Charles Lévesque, Christian Gagné, and Robert Sabourin. 2016. Bayesian Hyperparameter Optimization for Ensemble Learning. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence (UAI'16)*. AUAI Press, Arlington, Virginia, USA, 437–446.
- [229] Ke Li and Jitendra Malik. 2016. Learning to Optimize. *ArXiv* (June 2016). arXiv:cs.LG/1606.01885v1
- [230] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research* 18, 185 (2018), 1–52. <http://jmlr.org/papers/v18/16-558.html>
- [231] Liam Li and Ameet Talwalkar. 2019. Random Search and Reproducibility for Neural Architecture Search. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, Amir Globerson and Ricardo Silva (Eds.). AUAI Press, 129. <http://auai.org/uai2019/proceedings/papers/129.pdf>

- [232] Yu-Feng Li, Hai Wang, Tong Wei, and Wei-Wei Tu. 2019. Towards Automated Semi-Supervised Learning. *Proceedings of the AAAI Conference on Artificial Intelligence* 33 (jul 2019), 4237–4244. <https://doi.org/10.1609/aaai.v33i01.33014237>
- [233] Bin Liu. 2018. A Very Brief and Critical Discussion on AutoML. *ArXiv* (2018). arXiv:cs.AI/1811.03822v1
- [234] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. DARTS: Differentiable Architecture Search. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=S1eYHoC5FX>
- [235] Sijia Liu, Parikshit Ram, Deepak Vijaykeerthy, Djallel Bouneffouf, Gregory Bramble, Horst Samulowitz, Dakuo Wang, Andrew Conn, and Alexander Gray. 2020. An ADMM Based Framework for AutoML Pipeline Configuration. *Proceedings of the AAAI Conference on Artificial Intelligence* 34, 04 (apr 2020), 4892–4899. <https://doi.org/10.1609/aaai.v34i04.5926> arXiv:cs.LG/1905.00424v5
- [236] Zhengying Liu, Isabelle Guyon, Julio Jacques Junior, Meysam Madadi, Sergio Escalera, Adrien Pavao, Hugo Jair Escalante, Wei-Wei Tu, Zhen Xu, and Sebastien Treguer. 2019. AutoCV Challenge Design and Baseline Results. In *CAp 2019 - Conférence sur l'Apprentissage Automatique*. Toulouse, France. <https://hal.archives-ouvertes.fr/hal-02265053>
- [237] Ana C. Lorena, Aron I. Maciel, Péricles B. C. de Miranda, Ivan G. Costa, and Ricardo B. C. Prudêncio. 2017. Data complexity meta-features for regression problems. *Machine Learning* 107, 1 (dec 2017), 209–246. <https://doi.org/10.1007/s10994-017-5681-1>
- [238] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. 2017. The Expressive Power of Neural Networks: A View from the Width. In *Advances in Neural Information Processing Systems* 30, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 6231–6239. <http://papers.nips.cc/paper/7203-the-expressive-power-of-neural-networks-a-view-from-the-width.pdf>
- [239] Gang Luo. 2016. PredicT-ML: a tool for automating machine learning model building with big clinical data. *Health Information Science and Systems* 4, 1 (jun 2016). <https://doi.org/10.1186/s13755-016-0018-1>
- [240] Gang Luo. 2016. A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Network Modeling Analysis in Health Informatics and Bioinformatics* 5, 1 (may 2016). <https://doi.org/10.1007/s13721-016-0125-6>
- [241] David J. C. MacKay. 1992. A Practical Bayesian Framework for Backpropagation Networks. *Neural Computation* 4, 3 (may 1992), 448–472. <https://doi.org/10.1162/neco.1992.4.3.448>
- [242] Jorge G. Madrid, Hugo Jair Escalante, Eduardo F. Morales, Wei-Wei Tu, Yang Yu, Lisheng Sun-Hosoya, Isabelle Guyon, and Michele Sebag. 2019. Towards AutoML in the presence of Drift: first results. *ArXiv* (2019). arXiv:cs.LG/1907.10772v1
- [243] Francisco Madrigal, Camille Maurice, and Frédéric Lerasle. 2018. Hyper-parameter optimization tools comparison for multiple object tracking applications. *Machine Vision and Applications* 30, 2 (oct 2018), 269–289. <https://doi.org/10.1007/s00138-018-0984-1>
- [244] Brandon Malone, Kustaa Kangas, Matti Järvisalo, Mikko Koivisto, and Petri Myllymäki. 2017. AS-ASL: Algorithm Selection with Auto-sklearn. In *Proceedings of the Open Algorithm Selection Challenge (Proceedings of Machine Learning Research)*, Marius Lindauer, Jan N. van Rijn, and Lars Kotthoff (Eds.), Vol. 79. PMLR, Brussels, Belgium, 19–22. <http://proceedings.mlr.press/v79/malone17a.html>
- [245] Brandon Malone, Kustaa Kangas, Matti Järvisalo, Mikko Koivisto, and Petri Myllymäki. 2017. Empirical hardness of finding optimal Bayesian network structures: algorithm selection and runtime prediction. *Machine Learning* 107, 1 (dec 2017), 247–283. <https://doi.org/10.1007/s10994-017-5680-2>
- [246] Shaul Markovitch and Dan Rosenstein. 2002. Feature Generation Using General Constructor Functions. *Machine Learning* 49, 1 (2002), 59–98. <https://doi.org/10.1023/a:1014046307775>
- [247] Oded Maron and Andrew W. Moore. 1994. Hoeffding Races: Accelerating Model Selection Search for Classification and Function Approximation. In *Proceedings of the 6th International Conference on Neural Information Processing Systems (NIPS'93)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 59–66.
- [248] Christopher J Matheus and Larry A Rendell. 1989. Constructive Induction On Decision Trees. In *IJCAI*, Vol. 89. 645–650.
- [249] Michael McCloskey and Neal J. Cohen. 1989. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. In *Psychology of Learning and Motivation*. Elsevier, 109–165. [https://doi.org/10.1016/s0079-7421\(08\)60536-8](https://doi.org/10.1016/s0079-7421(08)60536-8)
- [250] Dale McConachie and Dmitry Berenson. 2020. Bandit-Based Model Selection for Deformable Object Manipulation. In *Springer Proceedings in Advanced Robotics*. Springer International Publishing, 704–719. https://doi.org/10.1007/978-3-030-43089-4_45 arXiv:cs.RO/1703.10254v1
- [251] Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. 2016. Towards Automatically-Tuned Neural Networks. In *Proceedings of the Workshop on Automatic Machine Learning (Proceedings of Machine Learning Research)*, Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (Eds.), Vol. 64. PMLR, New York, New York, USA, 58–65. http://proceedings.mlr.press/v64/mendoza_towards_2016.html
- [252] Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, Matthias Urban, Michael Burkart, Maximilian Dippel, Marius Lindauer, and Frank Hutter. 2019. Towards Automatically-Tuned Deep Neural Networks. In *Automated*

- Machine Learning*. Springer International Publishing, 135–149. https://doi.org/10.1007/978-3-030-05318-5_7
- [253] Ryszard S Michalski. 1986. *Understanding the nature of learning: Issues and research directions*. Morgan Kaufmann Publishers.
- [254] Mitar Milutinovic, Atılım Güneş Baydin, Robert Zinkov, William Harvey, Dawn Song, Frank Wood, and Wade Shen. 2017. End-to-end training of differentiable pipelines across machine learning frameworks. *NeurIPS Workshop* (2017).
- [255] J. Moćkus. 1975. On bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference Novosibirsk, July 1–7, 1974*. Springer Berlin Heidelberg, 400–404. https://doi.org/10.1007/3-540-07165-2_55
- [256] Felix Mohr, Marcel Wever, Eyke Hüllermeier, and Amin Faez. 2018. Towards the Automated Composition of Machine Learning Services. In *2018 IEEE International Conference on Services Computing (SCC)*. IEEE. <https://doi.org/10.1109/scc.2018.00039>
- [257] Felix Mohr, Marcel Wever, and Eyke Hüllermeier. 2018. Automated Machine Learning Service Composition. *ArXiv* (September 2018). arXiv:cs.SE/1809.00486v1
- [258] Felix Mohr, Marcel Wever, and Eyke Hüllermeier. 2018. ML-Plan: Automated machine learning via hierarchical planning. *Machine Learning* 107, 8-10 (jul 2018), 1495–1515. <https://doi.org/10.1007/s10994-018-5735-z>
- [259] Gholamreza Nakhaeizadeh and Alexander Schnabl. 1997. Development of Multi-Criteria Metrics for Evaluation of Data Mining Algorithms. In *KDD*. 37–42.
- [260] Fatemeh Nargesian, Horst Samulowitz, Udayan Khurana, Elias B. Khalil, and Deepak Turaga. 2017. Learning Feature Engineering for Classification. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization. <https://doi.org/10.24963/ijcai.2017/352>
- [261] Brady Neal, Sarthak Mittal, Aristide Baratin, Vinayak Tantia, Matthew Scicluna, Simon Lacoste-Julien, and Ioannis Mitliagkas. 2018. A Modern Take on the Bias-Variance Tradeoff in Neural Networks. *ArXiv* (2018). arXiv:cs.LG/1810.08591
- [262] P. Nguyen, M. Hilario, and A. Kalousis. 2014. Using Meta-mining to Support Data Mining Workflow Planning and Optimization. *Journal of Artificial Intelligence Research* 51 (nov 2014), 605–644. <https://doi.org/10.1613/jair.4377>
- [263] Tien-Dung Nguyen, Bogdan Gabrys, and Katarzyna Musial. 2020. AutoWeka4MCPS-AVATAR: Accelerating Automated Machine Learning Pipeline Composition and Optimisation. *ArXiv* (2020). arXiv:cs.LG/2011.11846v1
- [264] Tien-Dung Nguyen, Tomasz Maszczyk, Katarzyna Musial, Marc-André Zöller, and Bogdan Gabrys. 2020. AVATAR - Machine Learning Pipeline Evaluation Using Surrogate Model. In *Lecture Notes in Computer Science*. Springer International Publishing, 352–365. https://doi.org/10.1007/978-3-030-44584-3_28
- [265] Eleni Nisioti, K Chatzidimitriou, and A Symeonidis. 2018. Predicting hyperparameters from meta-features in binary classification problems. In *AutoML Workshop at ICML*.
- [266] Mustafa V. Nural, Hao Peng, and John A. Miller. 2017. Using meta-learning for model type selection in predictive big data analytics. In *2017 IEEE International Conference on Big Data (Big Data)*. IEEE. <https://doi.org/10.1109/bigdata.2017.8258149>
- [267] 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 2016 on Genetic and Evolutionary Computation Conference - GECCO '16*. ACM Press. <https://doi.org/10.1145/2908812.2908918>
- [268] Randal S. Olson, Ryan J. Urbanowicz, Peter C. Andrews, Nicole A. Lavender, La Creis Kidd, and Jason H. Moore. 2016. Automating Biomedical Data Science Through Tree-Based Pipeline Optimization. In *Applications of Evolutionary Computation*. Springer International Publishing, 123–137. https://doi.org/10.1007/978-3-319-31204-0_9
- [269] Francesco Orabona. 2014. Simultaneous Model Selection and Optimization through Parameter-free Stochastic Learning. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1116–1124. <http://papers.nips.cc/paper/5503-simultaneous-model-selection-and-optimization-through-parameter-free-stochastic-learning.pdf>
- [270] E. S. Page. 1954. Continuous Inspection Schemes. *Biometrika* 41, 1/2 (jun 1954), 100. <https://doi.org/10.2307/2333009>
- [271] Sinno Jialin Pan and Qiang Yang. 2010. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 10 (oct 2010), 1345–1359. <https://doi.org/10.1109/tkde.2009.191>
- [272] Antonio Rafael Sabino Parmezan, Huei Diana Lee, and Feng Chung Wu. 2017. Metalearning for choosing feature selection algorithms in data mining: Proposal of a new framework. *Expert Systems with Applications* 75 (jun 2017), 1–24. <https://doi.org/10.1016/j.eswa.2017.01.013>
- [273] Michael J. Pazzani. 1998. Constructive Induction of Cartesian Product Attributes. In *Feature Extraction, Construction and Selection*. Springer US, 341–354. https://doi.org/10.1007/978-1-4615-5725-8_21
- [274] R. D. Peng. 2011. Reproducible Research in Computational Science. *Science* 334, 6060 (dec 2011), 1226–1227. <https://doi.org/10.1126/science.1213847>
- [275] Johann Petrak. 2000. Fast subsampling performance estimates for classification algorithm selection. In *Proceedings of the ECML-00 Workshop on Meta-Learning: Building Automatic Advice Strategies for Model Selection and Method Combination*. 3–14.

- [276] Bernhard Pfahringer, Hilan Bensusan, and Christophe G Giraud-Carrier. 2000. Meta-Learning by Landmarking Various Learning Algorithms. In *ICML*. 743–750.
- [277] Dragoljub Pokrajac, Aleksandar Lazarevic, and Longin Jan Latecki. 2007. Incremental Local Outlier Detection for Data Streams. In *2007 IEEE Symposium on Computational Intelligence and Data Mining*. IEEE. <https://doi.org/10.1109/cidm.2007.368917>
- [278] Philipp Probst, Anne-Laure Boulesteix, and Bernd Bischl. 2019. Tunability: Importance of Hyperparameters of Machine Learning Algorithms. *Journal of Machine Learning Research* 20, 53 (2019), 1–32. <http://jmlr.org/papers/v20/18-444.html>
- [279] Philipp Probst, Marvin N. Wright, and Anne-Laure Boulesteix. 2019. Hyperparameters and tuning strategies for random forest. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 9, 3 (jan 2019). <https://doi.org/10.1002/widm.1301>
- [280] Yubin Qu. 2018. Impact of Hyper Parameter Optimization for Cross-Project Software Defect Prediction. *International Journal of Performance Engineering* (2018). <https://doi.org/10.23940/ijpe.18.06.p20.12911299>
- [281] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y. Ng. 2007. Self-taught learning: Transfer Learning from Unlabeled Data. In *Proceedings of the 24th international conference on Machine learning - ICML '07*. ACM Press. <https://doi.org/10.1145/1273496.1273592>
- [282] Esteban Real, Chen Liang, David So, and Quoc Le. 2020. AutoML-Zero: Evolving Machine Learning Algorithms From Scratch. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Hal Daumé III and Aarti Singh (Eds.), Vol. 119. PMLR, Virtual, 8007–8019. <http://proceedings.mlr.press/v119/real20a.html>
- [283] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. 2017. Large-Scale Evolution of Image Classifiers. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70 (ICML'17)*. JMLR.org, 2902–2911.
- [284] Matthias Reif, Faisal Shafait, and Andreas Dengel. 2012. Meta-learning for evolutionary parameter optimization of classifiers. *Machine Learning* 87, 3 (apr 2012), 357–380. <https://doi.org/10.1007/s10994-012-5286-7>
- [285] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. 2020. A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions. *ArXiv* (2020). arXiv:cs.LG/2006.02903v1
- [286] John R. Rice. 1976. The Algorithm Selection Problem. In *Advances in Computers*. Elsevier, 65–118. [https://doi.org/10.1016/s0065-2458\(08\)60520-3](https://doi.org/10.1016/s0065-2458(08)60520-3)
- [287] Silvia Riedel and Bogdan Gabrys. 2005. Hierarchical Multilevel Approaches of Forecast Combination. In *Operations Research Proceedings 2004*. Springer Berlin Heidelberg, 479–486. https://doi.org/10.1007/3-540-27679-3_59
- [288] Silvia Riedel and Bogdan Gabrys. 2007. Combination of Multi Level Forecasts. *The Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology* 49, 2 (jul 2007), 265–280. <https://doi.org/10.1007/s11265-007-0076-3>
- [289] Silvia Riedel and Bogdan Gabrys. 2007. Dynamic Pooling for the Combination of Forecasts generated using Multi Level Learning. In *2007 International Joint Conference on Neural Networks*. IEEE. <https://doi.org/10.1109/ijcnn.2007.4370999>
- [290] S. Riedel and B. Gabrys. 2009. Pooling for Combination of Multilevel Forecasts. *IEEE Transactions on Knowledge and Data Engineering* 21, 12 (dec 2009), 1753–1766. <https://doi.org/10.1109/tkde.2009.18>
- [291] Pablo Riera-Fernandez, Cristian R. Munteanu, Nieves Pedreira-Souto, Raquel Martín-Romalde, Aliuska Duardo-Sanchez, and Humberto Gonzalez-Diaz. 2011. Definition of Markov-Harary Invariants and Review of Classic Topological Indices and Databases in Biology, Parasitology, Technology, and Social-Legal Networks. *Current Bioinformatics* 6, 1 (mar 2011), 94–121. <https://doi.org/10.2174/157489311795222338>
- [292] David Roschewitz, Kurt Driessens, and Pieter Collins. 2018. Simultaneous Ensemble Generation and Hyperparameter Optimization for Regression. In *Communications in Computer and Information Science*. Springer International Publishing, 116–130. https://doi.org/10.1007/978-3-319-76892-2_9
- [293] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence* 1, 5 (may 2019), 206–215. <https://doi.org/10.1038/s42256-019-0048-x>
- [294] Dymitr Ruta and Bogdan Gabrys. 2002. A Theoretical Analysis of the Limits of Majority Voting Errors for Multiple Classifier Systems. *Pattern Analysis and Applications* 5, 4 (oct 2002), 333–350. <https://doi.org/10.1007/s100440200030>
- [295] Dymitr Ruta and Bogdan Gabrys. 2003. Set Analysis of Coincident Errors and Its Applications for Combining Classifiers. In *Pattern Recognition and String Matching*. Springer US, 647–671. https://doi.org/10.1007/978-1-4613-0231-5_25
- [296] Dymitr Ruta and Bogdan Gabrys. 2005. Classifier selection for majority voting. *Information Fusion* 6, 1 (mar 2005), 63–81. <https://doi.org/10.1016/j.inffus.2004.04.008>
- [297] Dymitr Ruta, Bogdan Gabrys, and Christiane Lemke. 2011. A Generic Multilevel Architecture for Time Series Prediction. *IEEE Transactions on Knowledge and Data Engineering* 23, 3 (mar 2011), 350–359. <https://doi.org/10.1109/tkde.2010.137>
- [298] Manuel Martin Salvador. 2017. *Automatic and adaptive preprocessing for the development of predictive models*. Ph.D. Dissertation. Bournemouth University.
- [299] Manuel Martin Salvador, Marcin Budka, and Bogdan Gabrys. 2016. Adapting multicomponent predictive systems using hybrid adaptation strategies with auto-weka in process industry. In *Workshop on Automatic Machine Learning*.

48–57.

- [300] Manuel Martin Salvador, Marcin Budka, and Bogdan Gabrys. 2016. Effects of Change Propagation Resulting from Adaptive Preprocessing in Multicomponent Predictive Systems. *Procedia Computer Science* 96 (2016), 713–722. <https://doi.org/10.1016/j.procs.2016.08.255>
- [301] Manuel Martin Salvador, Marcin Budka, and Bogdan Gabrys. 2016. Towards Automatic Composition of Multi-component Predictive Systems. In *Lecture Notes in Computer Science*. Springer International Publishing, 27–39. https://doi.org/10.1007/978-3-319-32034-2_3
- [302] Manuel Martin Salvador, Marcin Budka, and Bogdan Gabrys. 2017. Modelling Multi-Component Predictive Systems as Petri Nets. In *15th Annual Industrial Simulation Conference*, J. Kacprzyk and J. Owsinski (Eds.), Eurosis-ETL, 17 – 23. <http://eprints.bournemouth.ac.uk/30605/>
- [303] Manuel Martin Salvador, Marcin Budka, and Bogdan Gabrys. 2019. Automatic Composition and Optimization of Multicomponent Predictive Systems With an Extended Auto-WEKA. *IEEE Transactions on Automation Science and Engineering* 16, 2 (apr 2019), 946–959. <https://doi.org/10.1109/tase.2018.2876430>
- [304] A. L. Samuel. 1959. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development* 3, 3 (jul 1959), 210–229. <https://doi.org/10.1147/rd.33.0210>
- [305] A. L. Samuel. 1960. Some Moral and Technical Consequences of Automation—A Refutation. *Science* 132, 3429 (sep 1960), 741–742. <https://doi.org/10.1126/science.132.3429.741>
- [306] Roger Schaefer, Henning Müller, and Adrien Depeursinge. 2016. Optimized Distributed Hyperparameter Search and Simulation for Lung Texture Classification in CT Using Hadoop. *Journal of Imaging* 2, 2 (jun 2016), 19. <https://doi.org/10.3390/jimaging2020019>
- [307] Brandon Schoenfeld, Christophe Giraud-Carrier, Mason Poggemann, Jarom Christensen, and Kevin Seppi. 2018. Preprocessor Selection for Machine Learning Pipelines. *ArXiv* (October 2018). arXiv:cs.LG/1810.09942v1
- [308] Sebastian Schuchmann. 2019. *Analyzing the Prospect of an Approaching AI Winter*. Master’s thesis. <https://doi.org/10.13140/RG.2.2.10932.91524>
- [309] Steven L. Scott. 2010. A modern Bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry* 26, 6 (nov 2010), 639–658. <https://doi.org/10.1002/asmb.874>
- [310] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. 2015. Hidden Technical Debt in Machine Learning Systems. In *Advances in Neural Information Processing Systems* 28, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 2503–2511. <http://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf>
- [311] Frank Sehnke, Martin D Felder, and Anton K Kaifel. 2012. Learn-o-matic: A fully automated machine learning suite for profile retrieval applications. In *Proceedings of the 2012 EUMETSAT Meteorological Satellite Conference*.
- [312] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. 2010. Parameter-exploring policy gradients. *Neural Networks* 23, 4 (may 2010), 551–559. <https://doi.org/10.1016/j.neunet.2009.12.004>
- [313] Floarea Serban, Joaquin Vanschoren, Jörg-Uwe Kietz, and Abraham Bernstein. 2013. A survey of intelligent assistants for data analysis. *Comput. Surveys* 45, 3 (jun 2013), 1–35. <https://doi.org/10.1145/2480741.2480748>
- [314] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. 2016. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proc. IEEE* 104, 1 (jan 2016), 148–175. <https://doi.org/10.1109/jproc.2015.2494218>
- [315] Michael R. Smith, Logan Mitchell, Christophe Giraud-Carrier, and Tony Martinez. 2014. Recommending Learning Algorithms and Their Associated Hyperparameters. In *Proceedings of the 2014 International Conference on Meta-Learning and Algorithm Selection - Volume 1201 (MLAS’14)*. CEUR-WS.org, Aachen, DEU, 39–40.
- [316] Kate A. Smith-Miles. 2008. Cross-disciplinary perspectives on meta-learning for algorithm selection. *Comput. Surveys* 41, 1 (dec 2008), 1–25. <https://doi.org/10.1145/1456650.1456656>
- [317] Jasper Snoek, Hugo Larochelle, and R Adams. 2011. Opportunity cost in Bayesian optimization. In *NIPS Workshop on Bayesian Optimization, Sequential Experimental Design, and Bandits*.
- [318] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.), Vol. 25. Curran Associates, Inc., 2951–2959. <https://proceedings.neurips.cc/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf>
- [319] Carlos Soares, Johann Petrak, and Pavel Brazdil. 2001. Sampling-Based Relative Landmarks: Systematically Test-Driving Algorithms before Choosing. In *Progress in Artificial Intelligence*. Springer Berlin Heidelberg, 88–95. https://doi.org/10.1007/3-540-45329-6_12
- [320] Evan Randall Sparks. 2016. *End-to-End Large Scale Machine Learning with KeystoneML*. Ph.D. Dissertation. UC Berkeley.

- [321] Evan R. Sparks, Ameet Talwalkar, Daniel Haas, Michael J. Franklin, Michael I. Jordan, and Tim Kraska. 2015. Automating model search for large scale machine learning. In *Proceedings of the Sixth ACM Symposium on Cloud Computing - SoCC '15*. ACM Press. <https://doi.org/10.1145/2806777.2806945>
- [322] Evan R. Sparks, Shivaram Venkataraman, Tomer Kaftan, Michael J. Franklin, and Benjamin Recht. 2017. KeystoneML: Optimizing Pipelines for Large-Scale Advanced Analytics. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE. <https://doi.org/10.1109/icde.2017.109>
- [323] Pablo Sprechmann, Siddhant Jayakumar, Jack Rae, Alexander Pritzel, Adria Puigdomenech Badia, Benigno Uria, Oriol Vinyals, Demis Hassabis, Razvan Pascanu, and Charles Blundell. 2018. Memory-based Parameter Adaptation. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=rkfOvGbCW>
- [324] Christian Steinruecken, Emma Smith, David Janz, James Lloyd, and Zoubin Ghahramani. 2019. The Automatic Statistician. In *Automated Machine Learning*. Springer International Publishing, 161–173. https://doi.org/10.1007/978-3-030-05318-5_9
- [325] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. 2017. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM. <https://doi.org/10.1145/3071178.3071229>
- [326] Quan Sun, Bernhard Pfahringer, and Michael Mayo. 2012. Full model selection in the space of data mining operators. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion - GECCO Companion '12*. ACM Press. <https://doi.org/10.1145/2330784.2331014>
- [327] Quan Sun, Bernhard Pfahringer, and Michael Mayo. 2013. Towards a Framework for Designing Full Model Selection and Optimization Systems. In *Multiple Classifier Systems*. Springer Berlin Heidelberg, 259–270. https://doi.org/10.1007/978-3-642-38067-9_23
- [328] Lisheng Sun-Hosoya, Isabelle Guyon, and Michèle Sebag. 2018. ActivMetal: Algorithm Recommendation with Active Meta Learning. In *IAL 2018 workshop, ECML PKDD*. <https://hal.archives-ouvertes.fr/hal-01931262> Poster.
- [329] Lisheng Sun-Hosoya, Isabelle Guyon, and Michèle Sebag. 2018. Lessons learned from the AutoML challenge. In *Conférence sur l'Apprentissage Automatique 2018*. Rouen, France. <https://hal.inria.fr/hal-01811454>
- [330] 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. <https://doi.org/10.1109/bigdata.2017.8257923>
- [331] Kevin Swersky, Jasper Snoek, and Ryan P Adams. 2013. Multi-Task Bayesian Optimization. In *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2004–2012. <http://papers.nips.cc/paper/5086-multi-task-bayesian-optimization.pdf>
- [332] Yoshiki Takahashi, Masato Asahara, and Kazuyuki Shudo. 2018. A framework for searching a predictive model. In *SysML Conference*, Vol. 2018.
- [333] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. 2016. Automated parameter optimization of classification techniques for defect prediction models. In *Proceedings of the 38th International Conference on Software Engineering - ICSE '16*. ACM Press. <https://doi.org/10.1145/2884781.2884857>
- [334] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. 2019. The Impact of Automated Parameter Optimization on Defect Prediction Models. *IEEE Transactions on Software Engineering* 45, 7 (jul 2019), 683–711. <https://doi.org/10.1109/tse.2018.2794977>
- [335] Abhishek Thakur and Artus Krohn-Grimberghe. 2015. AutoCompete: A Framework for Machine Learning Competition. *ArXiv* (July 2015). arXiv:stat.ML/1507.02188v1
- [336] Janek Thomas, Stefan Coors, and Bernd Bischl. 2018. Automatic Gradient Boosting. *ArXiv* (July 2018). arXiv:stat.ML/1807.03873v2
- [337] William R. Thompson. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* 25, 3-4 (dec 1933), 285–294. <https://doi.org/10.1093/biomet/25.3-4.285>
- [338] 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 - KDD '13*. ACM Press. <https://doi.org/10.1145/2487575.2487629>
- [339] A. Truong, A. Walters, J. Goodsitt, K. Hines, C. B. Bruss, and R. Farivar. 2019. Towards Automated Machine Learning: Evaluation and Comparison of AutoML Approaches and Tools. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. 1471–1479. <https://doi.org/10.1109/ICTAI.2019.00209>
- [340] Athanasios Tsakonas and Bogdan Gabrys. 2012. GRADIENT: Grammar-driven genetic programming framework for building multi-component, hierarchical predictive systems. *Expert Systems with Applications* 39, 18 (dec 2012), 13253–13266. <https://doi.org/10.1016/j.eswa.2012.05.076>
- [341] Athanasios Tsakonas and Bogdan Gabrys. 2013. A fuzzy evolutionary framework for combining ensembles. *Applied Soft Computing* 13, 4 (apr 2013), 1800–1812. <https://doi.org/10.1016/j.asoc.2012.12.027>

- [342] John W. Tukey. 1962. The Future of Data Analysis. *The Annals of Mathematical Statistics* 33, 1 (mar 1962), 1–67. <https://doi.org/10.1214/aoms/1177704711>
- [343] Arijit Ukil, Ishan Sahu, Chetanya Puri, Ayan Mukherjee, Rituraj Singh, Soma Bandyopadhyay, and Arpan Pal. 2018. AutoModeling: Integrated Approach for Automated Model Generation by Ensemble Selection of Feature Subset and Classifier. In *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE. <https://doi.org/10.1109/ijcnn.2018.8489730>
- [344] Roman Vainshtein, Asnat Greenstein-Messica, Gilad Katz, Bracha Shapira, and Lior Rokach. 2018. A Hybrid Approach for Automatic Model Recommendation. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management - CIKM '18*. ACM Press. <https://doi.org/10.1145/3269206.3269299>
- [345] Suzanne van den Bosch. 2017. *Automatic feature generation and selection in predictive analytics solutions*. Master's thesis. Radboud University.
- [346] Wil M. P. van der Aalst. 1998. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems and Computers* 08, 01 (feb 1998), 21–66. <https://doi.org/10.1142/s0218126698000043>
- [347] Jan N. van Rijn, Salisu Mamman Abdulrahman, Pavel Brazdil, and Joaquin Vanschoren. 2015. Fast Algorithm Selection Using Learning Curves. In *Advances in Intelligent Data Analysis XIV*. Springer International Publishing, 298–309. https://doi.org/10.1007/978-3-319-24465-5_26
- [348] Joaquin Vanschoren. 2011. Meta-Learning Architectures: Collecting, Organizing and Exploiting Meta-Knowledge. In *Studies in Computational Intelligence*. Springer Berlin Heidelberg, 117–155. https://doi.org/10.1007/978-3-642-20980-2_4
- [349] Joaquin Vanschoren. 2018. Meta-Learning: A Survey. *ArXiv* (October 2018). arXiv:cs.LG/1810.03548v1
- [350] Joaquin Vanschoren, Pavel Brazdil, and Jörg-Uwe Kietz (Eds.). 2012. *5th Planning to Learn Workshop WS28 at ECAI 2012*. Citeseer.
- [351] Joaquin Vanschoren, Pavel Brazdil, Carlos Soares, and Lars Kotthoff (Eds.). 2014. *Meta-learning and Algorithm Selection Workshop at ECAI 2014*.
- [352] Bruno Veloso, João Gama, and Benedita Malheiro. 2018. Self Hyper-Parameter Tuning for Data Streams. In *Discovery Science*. Springer International Publishing, 241–255. https://doi.org/10.1007/978-3-030-01771-2_16
- [353] Ricardo Vilalta, Christophe G Giraud-Carrier, Pavel Brazdil, and Carlos Soares. 2004. Using Meta-Learning to Support Data Mining. *IJCSA* 1, 1 (2004), 31–45.
- [354] Luis von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. 2008. reCAPTCHA: Human-Based Character Recognition via Web Security Measures. *Science* 321, 5895 (aug 2008), 1465–1468. <https://doi.org/10.1126/science.1160379>
- [355] Dakuo Wang, Justin D. Weisz, Michael Muller, Parikshit Ram, Werner Geyer, Casey Dugan, Yla Tausczik, Horst Samulowitz, and Alexander Gray. 2019. Human-AI Collaboration in Data Science: Exploring Data Scientists' Perceptions of Automated AI. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (nov 2019), 1–24. <https://doi.org/10.1145/3359313>
- [356] G. Wang, Q. Song, H. Sun, X. Zhang, B. Xu, and Y. Zhou. 2013. A Feature Subset Selection Algorithm Automatic Recommendation Method. *Journal of Artificial Intelligence Research* 47 (may 2013), 1–34. <https://doi.org/10.1613/jair.3831>
- [357] Lidan Wang, Minwei Feng, Bowen Zhou, Bing Xiang, and Sridhar Mahadevan. 2015. Efficient Hyper-parameter Optimization for NLP Applications. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. <https://doi.org/10.18653/v1/d15-1253>
- [358] Weiyu Wang and Keng Siau. 2019. Artificial Intelligence, Machine Learning, Automation, Robotics, Future of Work and Future of Humanity. *Journal of Database Management* 30, 1 (jan 2019), 61–79. <https://doi.org/10.4018/jdm.2019010104>
- [359] Marcel Dominik Wever, Felix Mohr, and Eyke Hüllermeier. 2018. ML-Plan for Unlimited-Length Machine Learning Pipelines. In *ICML 2018 AutoML Workshop*.
- [360] N. Wiener. 1960. Some Moral and Technical Consequences of Automation. *Science* 131, 3410 (may 1960), 1355–1358. <https://doi.org/10.1126/science.131.3410.1355>
- [361] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. 2019. A Survey on Neural Architecture Search. *ArXiv* (2019). arXiv:cs.LG/1905.01392v2
- [362] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. 2015. Hyperparameter Search Space Pruning – A New Component for Sequential Model-Based Hyperparameter Optimization. In *Machine Learning and Knowledge Discovery in Databases*. Springer International Publishing, 104–119. https://doi.org/10.1007/978-3-319-23525-7_7
- [363] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. 2015. Learning hyperparameter optimization initializations. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE. <https://doi.org/10.1109/dsaa.2015.7344817>
- [364] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. 2016. Hyperparameter Optimization Machines. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE. <https://doi.org/10.1109/dsaa>

2016.12

- [365] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. 2017. Automatic Frankensteining: Creating Complex Ensembles Autonomously. In *Proceedings of the 2017 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, 741–749. <https://doi.org/10.1137/1.9781611974973.83>
- [366] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. 2017. Scalable Gaussian process-based transfer surrogates for hyperparameter optimization. *Machine Learning* 107, 1 (dec 2017), 43–78. <https://doi.org/10.1007/s10994-017-5684-y>
- [367] D.H. Wolpert and W.G. Macready. 1997. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1, 1 (apr 1997), 67–82. <https://doi.org/10.1109/4235.585893>
- [368] David H. Wolpert. 1996. The Lack of A Priori Distinctions Between Learning Algorithms. *Neural Computation* 8, 7 (oct 1996), 1341–1390. <https://doi.org/10.1162/neco.1996.8.7.1341>
- [369] Catherine Wong, Neil Houlsby, Yifeng Lu, and Andrea Gesmundo. 2018. Transfer Learning with Neural AutoML. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc., 8356–8365. <https://proceedings.neurips.cc/paper/2018/file/bdb3c278f45e6734c35733d24299d3f4-Paper.pdf>
- [370] Meng-Sung Wu and Jun-Yi Lu. 2018. Automated Machine Learning Algorithm Mining for Classification Problem. In *Machine Learning and Data Mining in Pattern Recognition*. Springer International Publishing, 380–392. https://doi.org/10.1007/978-3-319-96136-1_30
- [371] Mengwei Xu, Yuxin Zhao, Kaigui Bian, Gang Huang, Qiaozhu Mei, and Xuanzhe Liu. 2020. Federated Neural Architecture Search. *ArXiv* (2020). arXiv:cs.LG/2002.06352v4
- [372] Bing Xue, Mengjie Zhang, Will N. Browne, and Xin Yao. 2016. A Survey on Evolutionary Computation Approaches to Feature Selection. *IEEE Transactions on Evolutionary Computation* 20, 4 (aug 2016), 606–626. <https://doi.org/10.1109/tevc.2015.2504420>
- [373] Li Yang and Abdallah Shami. 2020. On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice. *Neurocomputing* (jul 2020). <https://doi.org/10.1016/j.neucom.2020.07.061>
- [374] Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. 2018. Taking Human out of Learning Applications: A Survey on Automated Machine Learning. *ArXiv* (October 2018). arXiv:cs.AI/1810.13306v4
- [375] Xin Yao. 1999. Evolving artificial neural networks. *Proc. IEEE* 87, 9 (1999), 1423–1447. <https://doi.org/10.1109/5.784219>
- [376] Yuanshun Yao, Zhujun Xiao, Bolun Wang, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. 2017. Complexity vs. performance: empirical analysis of machine learning as a service. In *Proceedings of the 2017 Internet Measurement Conference on - IMC '17*. ACM Press. <https://doi.org/10.1145/3131365.3131372>
- [377] Dani Yogatama and Gideon Mann. 2014. Efficient Transfer Learning Method for Automatic Hyperparameter Tuning. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Samuel Kaski and Jukka Corander (Eds.), Vol. 33. PMLR, Reykjavik, Iceland, 1077–1085. <http://proceedings.mlr.press/v33/yogatama14.html>
- [378] Chen Yu, Bojan Karlaš, Jie Zhong, Ce Zhang, and Ji Liu. 2019. AutoML from Service Provider’s Perspective: Multi-device, Multi-tenant Model Selection with GP-EI. In *Proceedings of Machine Learning Research (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri and Masashi Sugiyama (Eds.), Vol. 89. PMLR, 2829–2838. <http://proceedings.mlr.press/v89/yu19e.html>
- [379] Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. 2020. Evaluating The Search Phase of Neural Architecture Search. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=H1loF2NFwr>
- [380] Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. 2018. Towards Automated Deep Learning: Efficient Joint Neural Architecture and Hyperparameter Search. *ArXiv* (July 2018). arXiv:cs.LG/1807.06906v1
- [381] Xueqiang Zeng and Gang Luo. 2017. Progressive sampling-based Bayesian optimization for efficient and automatic machine learning model selection. *Health Information Science and Systems* 5, 1 (sep 2017). <https://doi.org/10.1007/s13755-017-0023-z>
- [382] Yuyu Zhang, Mohammad Taha Bahadori, Hang Su, and Jimeng Sun. 2016. FLASH: Fast Bayesian Optimization for Data Analytic Pipelines. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*. ACM Press. <https://doi.org/10.1145/2939672.2939829>
- [383] Indre Žliobaitė, Albert Bifet, Mohamed Gaber, Bogdan Gabrys, Joao Gama, Leandro Minku, and Katarzyna Musial. 2012. Next challenges for adaptive learning systems. *ACM SIGKDD Explorations Newsletter* 14, 1 (dec 2012), 48. <https://doi.org/10.1145/2408736.2408746>
- [384] Indrė Žliobaitė, Marcin Budka, and Frederic Stahl. 2015. Towards cost-sensitive adaptation: When is it worth updating your predictive model? *Neurocomputing* 150 (feb 2015), 240–249. <https://doi.org/10.1016/j.neucom.2014.05.084>

- [385] Indre Zliobaite and Bogdan Gabrys. 2014. Adaptive Preprocessing for Streaming Data. *IEEE Transactions on Knowledge and Data Engineering* 26, 2 (feb 2014), 309–321. <https://doi.org/10.1109/tkde.2012.147>
- [386] Barret Zoph and Quoc V. Le. 2017. Neural Architecture Search with Reinforcement Learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. <https://openreview.net/forum?id=r1Ue8Hcxg>
- [387] Marc-André Zöllner and Marco F. Huber. 2019. Benchmark and Survey of Automated Machine Learning Frameworks. *ArXiv* (April 2019). arXiv:cs.LG/1904.12054v2