Towards Interactive Curation & Automatic Tuning of ML Pipelines

Carsten Binnig[1,2] Benedetto Buratti[1] Yeounoh Chung[1] Cyrus Cousins[1] Tim Kraska[1,3]
Zeyuan Shang[1] Eli Upfal[1] Robert Zeleznik[1] Emanuel Zgraggen[1]
[1] Brown University, USA [2] TU Darmstadt, Germany [3] Massachusetts Institute of Technology, USA

ABSTRACT

Democratizing Data Science requires a fundamental rethinking of the way data analytics and model discovery is done. Available tools for analyzing massive data sets and curating machine learning models are limited in a number of fundamental ways. First, existing tools require well-trained data scientists to select the appropriate techniques to build models and to evaluate their outcomes. Second, existing tools require heavy data preparation steps and are often too slow to give interactive feedback to domain experts in the model building process, severely limiting the possible interactions. Third, current tools do not provide adequate analysis of statistical risk factors in the model development. In this work, we present the first iteration of *QuIC-M* (pronounced quick-m), an interactive human-in-the-loop data exploration and model building suite. The goal is to enable domain experts to build the machine learning pipelines an order of magnitude faster than machine learning experts while having model qualities comparable to expert solutions.

ACM Reference Format:
Carsten Binnig[1,2] Benedetto Buratti[1] Yeounoh Chung[1] Cyrus Cousins[1] Tim Kraska[1,3] Zeyuan Shang[1] Eli Upfal[1] Robert Zeleznik[1] Emanuel Zgraggen[1] . 2018. Towards Interactive Curation & Automatic Tuning of ML Pipelines. In *DEEM'18: International Workshop on Data Management for End-to-End Machine Learning, June 15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3209889.3209891

1 INTRODUCTION

Companies store customer and sales information, researchers collect data by running experiments, and application or website developers store interaction logs. But all this data is useless without the means to analyze it. Extracting actionable insights from data has been left to highly trained individuals who have strong mathematics and computer science skills. They have the background to query databases to create insightful reports and visualizations, develop statistical models and implement scalable infrastructures to process large and complex data. For example, it is common practice for corporations to employ teams of data scientists that assist stakeholders in finding qualitative, data-driven insights to inform possible business decisions. Having such a high-entry bar to data analysis however presents several challenges. First, it presents a major bottleneck. While research is trying to understand and promote visualization and data literacy and educational institutions are ramping up their data science curricula there is still a shortage of skilled data scientists. Second, and more importantly, restricting data analysis to those with a computational background creates an inequality. Small business owners without programming skills or research domains where computational background might not be as prevalent are at a disadvantage as they can not capitalize on the power of data.

We believe that there is an opportunity for tool builders to create systems for people who are domain experts but neither mathematicians, statisticians nor programmers. We introduce the first version of a system for **Qu**ality-aware **I**nteractive **C**uration of **M**odels, called **QuIC-M** (pronounced quick-m). Making sense of data is exploratory by nature, and demands rapid iterations and all but the simplest analysis tasks, require humans-in-the-loop to effectively steer the process. *QuIC-M* exposes the data exploration and model building workflow through a novel pen-and-touch interface allowing domain experts to seamlessly interleave data exploration steps with curation of machine learning pipelines. At the core of *QuIC-M* is an engine for automatic model discovery that takes a high-level problem specification and constraints from the user as input and automatically compiles a machine learning pipeline including the relevant pre-processing as well as model building steps. Through *QuIC-M* domain experts can thus build complex machine learning models at a fast pace without the need to involve a data scientist and without sacrificing quality.

Empowering novice users to automatically analyze data also comes with drawbacks since it exposes them to "the pitfalls that scientists are trained to avoid" [2]. We discussed these "pitfalls" and how to avoid them in recent related work [1, 4]. In this paper, we focus on *QuIC-M*'s architecture for automatic machine learning pipeline discovery.

2 OVERVIEW

To enable automatic model discovery, we propose a system which takes "problems" provided by domain experts as inputs and generates machine learning pipelines as outputs. Over time, the system adopts and improves its performance by learning from past problems. This section contains definitions and an overview of our architecture and the following sections provide details about individual parts of our approach.

A problem consists of a dataset and a target attribute for prediction. Domain experts can specify problems, and potential additional input such as constraints on which features to use or new user-defined features, through simple gestures in our pen-and-touch UI. Figure 1 shows the basic UI and the overall architecture of our system. A user can specify a problem through simple drag and drop gestures. In the example our user instructs the system to build a pipeline that predicts "hall of fame" using the features "home runs" and "batting average".

Based on the problem specification, our system will then automatically attempt to find and present a machine learning pipeline. While the goal of the system is to use signals in the data and enumerate over the different possible models to find the best pipeline, the human can focus on providing domain knowledge. To this end, the pipeline enumeration process is done progressively, where the system gradually optimizes over the space of possible pipelines, in order to display results early to users and allow them to provide

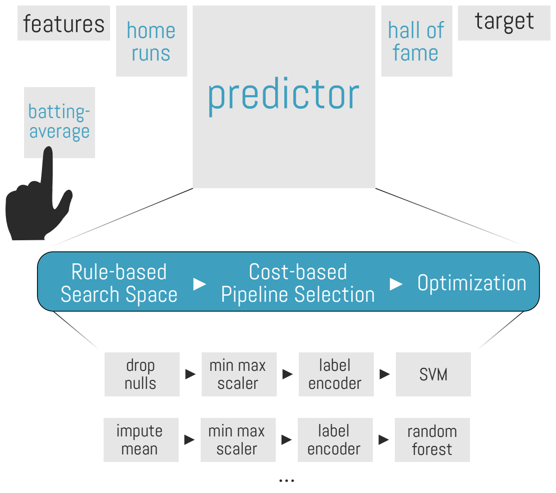

Figure 1: *QuIC-M*'s UI and architecture overview of automatic pipeline search.

feedback; e.g., by providing additional model constraints or visually changing decision planes based on domain knowledge.

During the design of our system we particularly focused on the following aspects. (1) *Progressiveness*: data analysis is an exploratory and iterative process, meaning that high-latencies are counterproductive and limit the rate at which domain experts can produce insights. Instead of waiting minutes or hours on results, we want users to see early results as soon as possible. We consciously trade-off some performance for interactivity by making all of our components output results in a progressive-fashion through incremental computation; (2) *User-steered*: we provide different opportunities for better user engagement, e.g., users may fix some parts of the pipeline and let the system fill others, and users could change the cost model to favor specific pipelines; (3) *Modularity*: usually there exist numerous implementations for each component in our system, by explicitly specifying the interfaces of each component, each implementation is interchangeable, thus making our system modular and highly configurable; (4) *Quality-awareness*: even with quality control techniques such as hold-outs and cross-validation, automatic tuning and incrementally refining a model increases the risk of finding a model which just works by chance. While not the focus of this paper, we are therefore investigating new safe learning to learn techniques.

For learning models based on more traditional ML techniques such as decision trees or SVMs, our system performs its automatic pipeline search based on three steps: *Rule-based Search Space*, *Cost-based Pipeline Selection* and *Optimization* and progressively and iteratively presents the currently best pipeline it has found back to the user. For deep learning based techniques, we are developing automatic tuning techniques along the lines of [5]. In this paper, we focus on the model composition using the more traditional ML techniques. The following sections explain these steps in detail.

3 RULE-BASED SEARCH SPACE

We use rules to create a search space of possible solutions. Rules create easy-to-explain solutions and allow us to extend the system by incorporating best practises from data scientists.

We denote a *pipeline* as an end-to-end solution to produce the predicted targets for a given problem. A pipeline consists of machine learning primitives (e.g., feature selection, pre-processing, model) and corresponding hyper-parameters. Given a specific problem, machine learning experts create pipelines based on their experiences, for example, for a classification problem, they tend to use classical models like SVMs. Based on this observation, we generate candidate pipelines through rules collected from best practices, which are gathered by analyzing hand-crafted solutions for existing problems from various sources such as Kaggle competitions [3] or the OpenML website [1]. To efficiently organize these candidate pipelines, we define the *search space* as the space for all possible candidate pipelines. Note that these best practice rules are usually parametric, which means that they can be further fine-tuned with feedback from interactions. Therefore the search space can be evolved through time, from a general range of possible answers to more problem-specific.

Based on the definition of *pipeline*, we know that a pipeline consists of two parts: (1) the structure of a pipeline, i.e., what primitives are in the pipeline (e.g., label encoding, min-max scaling, SVM, Random Forest) and how they are connected to form the pipeline; (2) the hyper-parameters of these primitives. We define a *pipeline arm* as a cluster of pipelines that share the same primitives but have different hyper-parameters. Then the search space consists of lots of pipeline arms.

Based on the definitions above, we propose three types of rules: (1) *structure rule*, which generates the primitives in the pipeline; (2) *parameter rule*, which specifies how to generate hyper-parameters for each primitive; (3) *enforcement rule*, which checks that the generated pipeline fulfills any requirements (e.g., categorical features have to be encoded to numerical values). For each rule, there is a corresponding condition specifying what kinds of requirements should be fulfilled to apply this rule (e.g., you can only use the Random Forest classifier for classification problem), and the system would only apply applicable rules to build the search space. Users are also able to provide some hints to prioritize between rules, for example, users may specify that they would like to use SVM first.

Structure Rules propose the possibly useful primitives for a specific problem (e.g., classification, regression, collaborative filtering) or dataset schema (e.g. apply one hot encoding for categorical features). For now, we have integrated more than 20 structure rules, including how to encode categorical features, how to scale numerical values, how to choose model for classification, regression and collaborative filtering problems, how to extract features from raw text and images, how to construct the graph for graph dataset. Our system supports classical classification, regression, collaborative filtering and community detection problems, and supports processing raw data like text, images and graphs.

Parameter Rules generate reasonable distributions for hyper-parameters. We support uniform and log uniform distributions for integer and float values, and uniform distribution for categorical values for hyper-parameters. For example, the system may choose *linear*, *poly*, *sigmoid* or *rbf* for kernel of SVM, and choose a log uniform distribution for the regularization factor λ. We also support conditions between hyper-parameters, for example, the *hinge* loss and *l1* norm for the penalization of the linear SVM can not be used together.

[1]https://www.openml.org/

Enforcement Rules check the pipeline to make sure it is valid. For example, all the categorical features should be encoded into numerical values; raw data should be featurized.

In the future, we are going to add more rules into the system to make it support more problems and data types (e.g., time series, sounds), whereas our design makes it flexible to add new rules; we also plan to make rules learned over time, for example, adjusting the ranges of hyper-parameters based on previous iterations. Besides, we will investigate how to generate rule automatically and prioritize rules by analyzing best pipelines for some seed problems.

4 COST-BASED PIPELINE SELECTION

Considering the complexity of machine learning problems and pipelines, the search space will be huge. To this end, we devise a cost model to measure the "promisingness" of a pipeline. By filtering out pipelines with high cost, we could prune the search space, saving resources and reducing the overall search latency. For now, we consider four factors of a pipeline in the cost model: (1) *time* for training and testing the pipeline; (2) *quality* of a pipeline, for example, the accuracy of classification; (3) *quality gain* of a pipeline, that is, how much quality this pipeline could potentially increased based on the last best; (4) *risk* of a pipeline, that is, the variance of quality. These factors will be combined together by different weights. For example, at first, the time is important because the user would like to see a workable pipeline as fast as possible, then the quality may matter much more in later iterations.

To build the cost model, we use two methods: (1) *prior knowledge*, for example, a Stochastic Gradient Descent (SGD) based linear classifier will be much faster in general than a SVM, with a little worse quality; (2) *history*, we could use the results of validated pipelines in previous iterations to estimate new pipelines. With more iterations and more pipelines validated, the history will be more useful and the cost model will become more accurate. In other words, the cost models can be learned as well.

5 OPTIMIZATION ALGORITHMS

The pruned *search space* $\mathcal{X}$ obtained by the rule-based model contains all the possible candidate pipelines that are significant for the task given by the user. This search space is received as an input by the optimizer. Then consider a **fitness function** f which evaluates a pipeline on a sample S returning a real value $f_S : \mathcal{X} \to R$. The returned value can be an arbitrary metric that we want to use to evaluate the pipelines. Finally let's define x^* as the best pipeline, that is $x^* = argmax_{x \in \mathcal{X}} f_S(x)$ and $y^* = f_S(x^*)$ as the score that pipeline x^* obtains on this task S. In other terms y^* is the best possible obtainable performance for task S.

$\mathcal{X}$ is a highly heterogeneous space, with real, discrete and categorical features with conditional dependencies among them (e.g., SVM's γ and c have no meaningful interpretation under Random Forest). Additionally, we do not have any information about the analytical form of f_S so we want to use it as an oracle by evaluating the pipelines performance and perform black-box optimization.

Here are several ways to perform this optimization:

- **Model-Free:** the optimizer probes the oracle function f_S with some sampling and evaluation strategy that do not rely on a specific model. Examples in this category are: Grid Search, Random Search, Multi-armed Bandits.

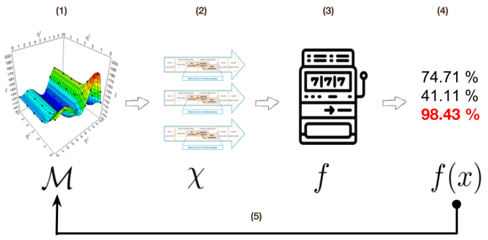

Figure 2: Optimization loop: (1) surrogate model, (2) sampled pipelines, (3) multi-armed bandits early termination strategy, (4) evaluation results, (5) surrogate model update

- **Model-Based:** the optimizer uses a surrogate model $\mathcal{M}_{f_S}$ of f_S built and updated out of the evaluation of the oracle itself. The surrogate model is designed to be treatable replacement of the original function, which is used to adaptively explore the search space. Examples in this category are: Cost-Based Pipeline Selection, Bayesian Optimization, Simulated Annealing.

The first two strategies are not in conflict with each other. As shown in figure 2 we can use model-based techniques to achieve the global optimum x^* sampling n most promising pipelines from $\mathcal{M}_{f_S}$ and use bandits to optimally allocate the budget by enforcing early termination during the evaluation stage. In order to accommodate the needs of the end user, the optimizer needs to provide both strong anytime results as well as very good final accuracy. This allows *QuIC-M* to be interactive while eventually delivering the best possible pipeline.

We designed Adaptive Pipeline Selection (APS), a "multi-armed bandits"-like early termination strategy to sub-optimally allocate an evaluation budget to m pipelines (arms). We split the data into independent train and validation sets and randomly sample m pipelines from $\mathcal{X}$. Additionally, we split the train set into smaller samples of increasing size. The APS algorithm gets an input training set S_t, a validation set S_v, number of sub-epochs n and number of meta-epochs N. Adaptive Selection Algorithm works over two time-scale phases: n sub-epochs and N meta-epochs .

- A **sub-epochs** is a training phase in which the sampled m pipelines train on a subset of the whole dataset. At the i-th sub-epoch we have survival m^i pipelines that are going to train on $\frac{|S| \cdot i}{n}$ samples.
- A **meta-epoch** is composed by a sequence of n sub-epochs. After a whole meta-epoch the survival pipelines have used the whole training set at least once.

At the end of each sub-epoch we are apply to the m^i pipelines the following halting criteria:

Halting Criteria 1. *At sub-epoch i, $\forall m^i_j \in m^i$ pipelines train them on a sub-sample of S of size $\frac{|S| \cdot i}{n}$ and compute their train error. If the pipeline m^j training error is above the best validation error seen so far, terminate it.*

Our assumption here is that the training accuracy can be safely considered as a reasonable upper bound to the best validation accuracy achievable with the pipeline; thus, we halt pipelines of which

Algorithm 1: Adaptive Pipeline Selection

```
input : meta_num, sub_num, arms_num, X_train,
        y_train
1  interval_size = int(ceil(X_train.shape[0]/sub_num))
2  pips = []
3  kills = 0
   foreach meta_epoch ∈ range(meta_num) do
4      X_valid, y_valid = get_fresh_validation_data()
5      pips = sample_pipelines_adaptively(pips, arms_num)
       foreach
         $start \in range(0, X\_train.shape[0], interval\_size)$: do
6          if start + interval_size < X_train.shape[0] then
7              end = start + interval_size
           else
8              end = X_train.shape[0]
9          X_train_sub = X_train[0: end]
10         y_train_sub = y_train[0: end]
           foreach pip ∈ pips do
11             pip.fit(X_train_sub, y_train_sub)
12             train_score, valid_score = pip.score(X_train_sub,
                 y_train_sub, X_valid, y_valid)
               if valid_score > best_valid_score then
                   best_valid_score = valid_score
13             if train_score < best_valid_score then
14                 pips.remove(pip)
15                 kills += 1
16 return best_pipeline(pips)
```

training accuracy drops below the best validation accuracy seen thus far. A meta-epoch ends in two cases: when its sub-epochs routines use the whole training set or no pipeline have survived to the previous halting criteria. In the latter case, we apply the following stopping rule:

Halting Criteria 2. *At sub-epoch i, if no pipeline survived from the previous sub-epoch $i-1$, abort the whole meta-epoch.*

It is important to notice that during the last n sub-epoch the survival m pipelines are trained on the whole dataset. At the end of each meta-epoch, we compute the validation score for each pipeline and update the surrogate model with those results. At this point, we start a new meta-epoch by adaptively sampling new n pipelines from $\mathcal{X}$ based on the updated surrogate model a new additional independent validation dataset. The validation data independence is crucial, because otherwise our surrogate model would overfit to a specific validation set. In this way APS can directly influence the evaluations running time by controlling the amount of data that is given to the pipelines. This is crucial in a distributed setting, where intra-cluster communication is usually the main bottleneck. By reducing the amount of distributed we can mitigate this problem.

6 PRELIMINARY RESULTS

To demonstrate the advantages of our system, we implemented a prototype with above mentioned techniques, the source code is at https://gitlab.com/BrownBigData/IDEA. We collected 85 datasets

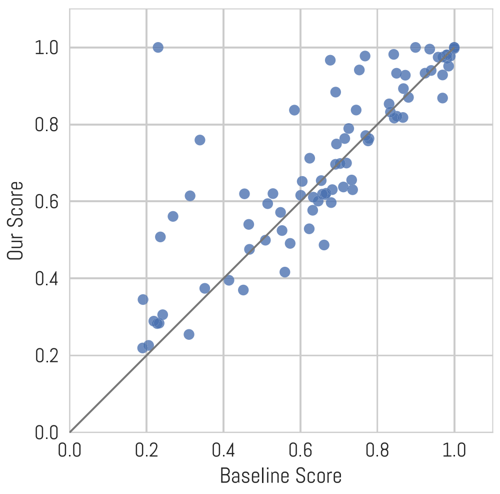

Figure 3: Comparison of F1 macro scores between hand-crafted (Baseline Score) and automatically found classification pipelines using *QuIC-M* over 85 datasets.

from OpenML [2] , allocated 300 seconds running time for each dataset, and compared best pipelines found by our system with hand-crafted solutions. As shown in Figure 3 the current implementation of our system is able to generate solutions for classification tasks which are on par with hand-crafted solutions.

7 CONCLUSION AND FUTURE WORK

In this paper, we propose *QuIC-M*, an interactive data exploration system for automatic searching and tuning of machine learning pipelines. *QuIC-M* used best practice rules to generate the search space of pipelines, and employed cost models to select promising pipelines. Further, we devised an adaptive pipeline selection algorithm to traverse this search space. Preliminary results showed that *QuIC-M* can find solutions comparable to hand-crafted ones.

We plan to extend our prototype to support more problems, include recent reinforcement learning techniques for automatically finding neural net architectures and to provide better quality guarantees for the pipelines that it finds. Moreover, we are going to extensively benchmark our system to understand more about the differences between our system and hand-crafted solutions and to run user studies to evaluate the user interface aspect of our system.

REFERENCES

[1] Andrew Crotty, Alex Galakatos, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. 2015. Vizdom: interactive analytics through pen and touch. *Proceedings of the VLDB Endowment* 8, 12 (2015), 2024–2027.
[2] Danyel Fisher, Rob DeLine, Mary Czerwinski, and Steven Drucker. 2012. Interactions with big data analytics. *interactions* 19, 3 (2012), 50–59.
[3] Mark Senn. 2017 (accessed December 29, 2017). *Kaggle Competitions.* https://www.kaggle.com/competitions
[4] Zheguang Zhao, Lorenzo De Stefani, Emanuel Zgraggen, Carsten Binnig, Eli Upfal, and Tim Kraska. 2017. Controlling false discoveries during interactive data exploration. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 527–540.
[5] Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).

[2]https://www.openml.org/