PipelineProfiler: A Visual Analytics Tool for the Exploration of AutoML Pipelines

Jorge Piazentin Ono, Sonia Castelo, Roque Lopez, Enrico Bertini, Juliana Freire, Claudio Silva
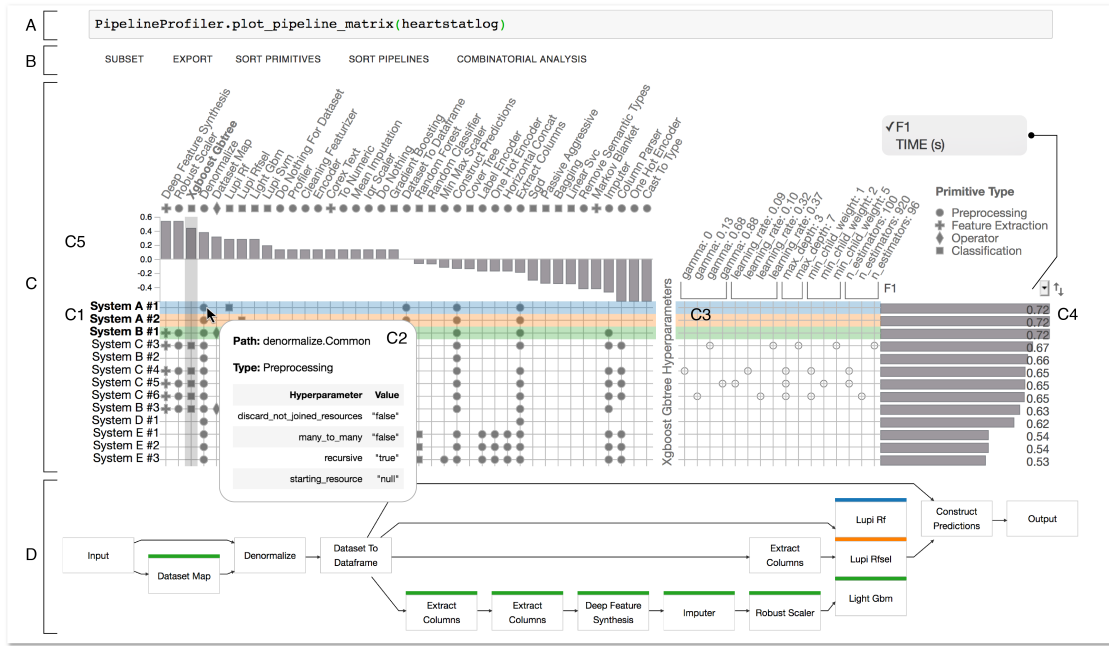
Fig. 1. *PipelineProfiler* applied to the analysis of binary classification pipelines generated by five AutoML systems for the *Statlog (Heart) Data Set*. A) The system is integrated with Jupyter Notebook and can be used with one line of code. B) *PipelineProfiler* menu, with options to subset, export, sort and perform automated analysis on pipelines. C) Pipeline Matrix: C1) Primitives (columns) used by the pipelines (rows). C2) Tooltip showing the metadata and hyperparameters for a primitive. C3) One-hot-encoded hyperparameters (columns) for the primitive *Xgboost Gbtree* across pipelines (rows). C4) Pipeline scores: users can select different metrics to rank pipelines. C5) Primitive Contribution View, showing the correlations between primitive usage and pipeline scores – here, we see that *Deep Feature Synthesis* has the highest correlation score with F1 scores. D) Pipeline Comparison View: visual comparison of the top-3 scoring pipelines (blue, orange and green).

Abstract— In recent years, a wide variety of automated machine learning (AutoML) methods have been proposed to search and generate end-to-end learning pipelines. While these techniques facilitate the creation of models for real-world applications, given their black-box nature, the complexity of the underlying algorithms, and the large number of pipelines they derive, it is difficult for their developers to debug these systems. It is also challenging for machine learning experts to select an AutoML system that is well suited for a given problem or class of problems. In this paper, we present the *PipelineProfiler*, an interactive visualization tool that allows the exploration and comparison of the solution space of machine learning (ML) pipelines produced by AutoML systems. *PipelineProfiler* is integrated with Jupyter Notebook and can be used together with common data science tools to enable a rich set of analyses of the ML pipelines and provide insights about the algorithms that generated them. We demonstrate the utility of our tool through several use cases where *PipelineProfiler* is used to better understand and improve a real-world AutoML system. Furthermore, we validate our approach by presenting a detailed analysis of a think-aloud experiment with six data scientists who develop and evaluate AutoML tools.

Index Terms—Automatic Machine Learning, Pipeline Visualization, Model Evaluation

✦

1 INTRODUCTION

Machine Learning (ML) has been successfully adopted in a plethora of applications. However, assembling end-to-end ML pipelines is a

All authors are with the New York University. E-mails: jorgehpo, s.castelo, rlopez, enrico.bertini, juliana.freire, csilva@nyu.edu

difficult endeavor that requires a time-consuming, trial-and-error process. This is difficult for ML experts and out of reach for subject-matter experts with little or no training in ML or computer science. AutoML systems have been proposed to address this challenge. Given a ML problem, AutoML aims to automate the synthesis of ML pipelines that perform well for the problem by searching over a space of possible pipelines which use different combinations of computational steps (or primitives) and values for their associated hyperparameters [28]. AutoML is having a substantial impact by making data scientists more efficient, enabling researchers to work on harder problems, and democratizing ML to less experienced developers [16, 21, 22].

Several open-source AutoML systems are currently available. For example, Auto-sklearn [22] creates pipelines based on a pool of 33 pre-processing and classification primitives, which contain 110 hyperparameters. In the context of the DARPA Data-Driven Discovery of Models (D3M) program [19], a collaborative effort involving multiple research groups, twenty AutoML systems were developed that synthesize pipelines from a pool with over 300 primitives [11, 13]. Because exploring the entire space of primitives and hyperparameter values is not feasible in practice, these systems use sophisticated search strategies to prune the space and reduce the number of pipelines they need to evaluate. For example, AutoWeka uses Bayesian optimization [48], Auto-sklearn uses meta-learning [22], TPOT uses genetic programming [37], and AlphaD3M uses deep learning [16].

Challenges in Understanding and Comparing AutoML Systems. Given the complexity of these systems, two important challenges arise. First, *it is difficult to evaluate their performance*. AutoML users may not trust automatically generated pipelines, as they do not understand the inner-workings of the blackboxes that generate and evaluate them [53]. To debug an AutoML system, developers must analyze logs consisting of synthesized pipeline instances and their outcomes. They need to assess, for instance, the efficiency of the search process, the structural diversity of the derived pipelines, how well the search covers the available primitives, and whether primitives were used correctly. This requires involved analyses that are further complicated due to the fact that not only do these logs contain a large number of instances, but the instances can have a complex structure and use a wide variety of primitives.

The second challenge emerges from the *need to compare different AutoML systems*. Given the growing number of available systems [16, 22, 25, 29, 35, 37], for users faced with the task of selecting a system, it is important to understand how well the systems perform for different problems, and identify features that contribute to a system being more or less effective than others for a task. Insights obtained in such a comparison are also of great value for AutoML developers, as these may help them improve their systems.

Visual analytics techniques have been proposed to make AutoML more transparent by enabling the exploration of the produced pipelines, the primitives employed, and their associated hyperparameters [7, 24, 38, 53, 56]. However, these have important limitations with respect to the challenges we outlined: (1) because their graphical encodings for pipelines only support fixed templates, they cannot explore pipelines that have complex structure; and (2) they do not support the comparison of multiple AutoML systems. For example: AutoAIViz [56] only supports sequential pipelines that consist of three primitives: two transformers (e.g., pre-processing or feature selection) and one estimator (e.g., classification or regression); ATMSeer [53] presents information only for the estimator step of the pipeline. Using these approaches, it is not possible to explore pipelines that have a complex structure such as directed acyclic graphs (DAG), or long pipelines that contain multiple estimators, such as the ones used for ensemble models which have been shown to outperform simpler pipelines [22].

It is worth noting that each of these techniques is tightly integrated with a specific AutoML system. For example, while ATMSeer [53] visualizes pipelines produced by ATM [45], AutoAIViz can only show pipelines from AutoAI [52]. They were were designed to explore the pipelines produced by a given AutoML system and they do not address the challenges involved in comparing different systems.

Our Approach. We propose *PipelineProfiler*, a visual analytics tool that enables the exploration and comparison of end-to-end ML pipelines produced by multiple AutoML systems. Fig. 1 shows the main components of the system. The Pipeline Matrix provides a visual summary of a collection of AutoML pipeline instances that captures structural information, including the primitives and associated hyperparameters, used in the pipelines, as well as the outcome of the pipeline encoded in a score. This representation is compact and can effectively encode pipelines derived by multiple AutoML systems that have both complex structure and use a variety of primitives. By showing the correlations between primitive usage and pipeline scores and how much primitives contribute to the score, the representation can also help uncover in-

sights into the suitability (or effectiveness) of primitives for specific data and problem types. Users can drill down into the structural details of pipelines, and examine both their differences and similarities. As we discuss later, these analyses enable the identification of patterns which can, for example, expose interesting aspects of the search strategies used by AutoML systems. *PipelineProfiler* is integrated with Jupyter Notebooks. This enables complex analyses to be performed over pipeline collections that leverage the rich ecosystem of Python tools for data science. In addition, because these analyses are scripted, they can be reproduced and re-used.

Our design was inspired by requirements from developers of AutoML systems and experts that evaluate and compare these systems in the context of the DARPA D3M project [19]. These experts used *PipelineProfiler* and provided feedback throughout its development. We demonstrate the effectiveness of *PipelineProfiler* through two case studies that highlight some of the actionable insights these experts derived while using the system. We also describe a think-aloud experiment, where six experts used our tool with a dataset and problem of their choice and described both insights they obtained and how the tool integrates with their existing evaluation and debugging workflows.

Our main contributions can be summarized as follows:

- We propose *PipelineProfiler*, which is, to the best of our knowledge the first visual analytics tool for exploring and comparing pipelines produced by multiple AutoML systems.

- The tool introduces new visual representations and interactions, including: the Pipeline Matrix, a visual representation that effectively summarize the pipelines, primitives and hyperparameter values produced by AutoML systems; and the Graph Comparison View, which applies graph matching to highlight the structural differences and similarities among a set of pipelines of interest.

- To derive the information needed to convey the correlation between primitives and pipeline scores, we propose a new algorithm that identifies groups of primitives that have a strong impact on pipeline scores.

- To demonstrate the effectiveness of *PipelineProfiler*, we present two use cases that show how the tool was used to obtains insights that resulted in efficiency improvements to an open-source AutoML system. We also discuss a detailed analysis of a think-aloud experiment with six data scientists, which suggests that the tool is both usable and useful, and highlights the benefits of the integration of the tool with Jupyter Notebooks both in the flexibility it affords for complex analyses and the simplicity in incorporating the tool into the experts' development and evaluation workflows.

We review related work in Section 2. In Section 3, we discuss the desiderata for the understanding and comparison of AutoML systems distilled from a set of interviews that we conducted with AutoML developers. We also present *PipelineProfiler* and describe how its different components were designed informed by these desiderata. The case studies and analysis of the think-aloud interviews are discussed in Section 4. We conclude in Section 5, where we outline directions for future work.

2 RELATED WORK

AutoML has emerged as an approach to simplify the use of ML for different applications, and many systems that support AutoML are currently available [16, 22, 25, 28, 29, 35, 37]. While early work on AutoML focused on hyperparameter optimization and on ML primitives, recent approaches aim to efficiently automate the synthesis of end-to-end pipelines – from data loading, pre-processing, feature extraction, feature selection, model fitting and selection, and hyper-parameter tuning [16, 22, 37, 42, 45]. AlphaD3M uses deep learning to learn how to incrementally construct ML pipelines, framing the problem of pipeline synthesis for model discovery as a single-player game with a neural network sequence model and Monte Carlo Tree Search (MCTS) [16]. Auto-sklearn produces pipelines using Bayesian optimization combined with meta-learning [22], and TPOT uses genetic programming and tree-based optimization [37].

During the search for well-performing pipelines, AutoML systems can generate a large number of pipelines. This makes the analysis of the search results a challenging problem, in particular when pipelines have similar scores. As a consequence, the selection of the best performing end-to-end pipeline becomes expensive, time-consuming and a tedious process. In the following, we discuss research that has attempted to tackle this challenges through visualization.

Explaining AutoML. The black box nature of AutoML systems and the difficulty in understanding the inner-workings of these systems lead to reduced trust in the pipelines they produce [53]. There have been some attempts to make the AutoML process more transparent through insightful visualizations of the resulting pipelines. These can be roughly grouped into two categories: hyperparameter visualization [24, 38, 54] and pipeline visualization [7, 56].

ATMSeer [53] and Google Vizier [24] are approaches to visualize hyperparameters. ATMSeer, which is integrated with the ATM AutoML framework [45], displays the predictive model (i.e., last step of the ML pipeline) together with its hyperparameters and performance metrics to the user. Users can use crossfiltering (e.g., over ML algorithms) to facilitate the exploration of large collection of pipelines and refine the search space of the AutoML system, if needed. Google Vizier makes use of a parallel coordinate view where hyperparameters and objective functions are displayed to the users. It allows them to examine how the different hyperparameters (dimensions) co-vary with each other and also against the objective function. Although these methods help AutoML users to analyze the generated pipelines, most of them only support the analysis of its last step which is the fitted model, leaving aside important aspects of the pipeline such as data cleaning and feature engineering. In contrast, *PipelineProfiler* provides a visualization to explore and analyze the end-to-end pipeline – from data ingestion and engineering, to model generation.

Systems that support pipeline visualization include AutoAIViz [56] and REMAP [7]. AutoAIViz uses Conditional Parallel Coordinates (PCP) [55] to represent sequential pipelines and their hyperparameters. The system provides a hierarchical visualization which shows the pipeline steps (on the first level of PCP) as well as the hyperparameters of each step (on the second level of PCP). REMAP [7] focuses on pipelines that use deep neural networks. It proposes a new glyph, called Sequential Neural Architecture Chips (SNAC), which shows both the layer type and dimensionality, and allows users to interactively add or remove layers from the networks. Despite their ability to show end-to-end pipelines, both systems can only show linear pipelines, making it difficult to explore pipelines created by different AutoML systems that have more complex structure. Furthermore, REMAP was designed specifically to visualize neural network architectures, and thus it is not suitable to explore general ML pipelines that use different learning techniques. In this work, our goal is to allow users to explore, compare and analyze pipelines generated by multiple AutoML systems which can have nonlinear pipeline structures and use a variety of primitives and learning techniques.

Visual Analytics for Model Selection. Selecting a *good* model among the potentially large set of models (or pipelines) derived by an AutoML system is a challenging problem that has attracted significant attention in the literature. Visual analytics systems such as Visus [41], TwoRavens [23], and Snowcat [6] provide a front-end to AutoML systems and guide subject-matter experts without training in data science through the model selection process. They focus on providing explanations for models, and some provide simple mechanisms to compare models (e.g., based on the scores, or the actual explanations). Other approaches focus exclusively on model selection. RegressionExplorer [15] enables the creation, evaluation and comparison of logistic regression models using subgroup analysis. Square [39] introduces a novel encoding to investigate different models by visually comparing multiple histograms based on the statistical performance metrics of a multi-class classifier. It also shows instance-level distribution information. Similarly, ModelTracker [2], an interactive instance-based visualization, enables multi-scale analysis. It visualizes predictions supporting both aggregate and instance-level performance information while enabling direct inspection of the data. The majority of these

methods was designed to evaluate and select predictive models based on the performance results. However, they do not take into account additional metrics like *running time*, i.e., how long pipelines take to run, or *primitive usage*, i.e., whether primitives are used correctly and effectively. *PipelineProfiler* not only encodes this information in a compact visual representation, but it also provides a usable interface that allows users interact with a pipeline collection at different levels of abstraction – from a high-level overview to drilling down to inspect details of select pipelines.

Interactive Model Steering. Systems such as TreePOD [36], BEAMES [14] and EnsembleMatrix [46] support the analysis and refinement of models through interactive visualizations, and allow users to explore the effects of modifying some parameters. BEAMES [14] lets users steer the training of new regression models from a set of previous models. It presents model performance information to the users and trains new models based on the user feedback. Users can modify feature/sample importance, and combine multiple models to create ensembles. EnsembleMatrix [46] enables users to steer the creation of ensemble decision tree models. With EnsembleMatrix, users can combine and choose weights for decision trees and interactively evaluate the performance of the ensemble model. TreePOD [36] supports the creation of decision trees with multiple objectives, including performance and interpretability. Users can look at the optimization procedure and guide it so that simpler solutions are found. These systems frequently support model steering by letting users try different settings during the model construction process. Although, they help users understand the impact of these parameters over the models, they do not consider other relevant steps (also called primitives) that are part of end-to-end pipelines like data ingestion and feature engineering, which could have a significant impact in the final model performance. The primitives contribution view in *PipelineProfiler* displays the correlations between primitive usage and scores, allowing users to infer which primitives can lead to well-performing pipelines. Users can then drill down and further explore individual pipelines, their primitives and hyperparameters.

3 *PipelineProfiler*: EXPLORING END-TO-END ML PIPELINES

In this section, we describe *PipelineProfiler*, a tool that enables the exploration of end-to-end machine learning pipelines produced by AutoML systems. We first present the desiderata we distilled from interviews with AutoML experts and subsequently used to guide our design choices. Then, we describe the components of *PipelineProfiler*, how they are integrated, and the algorithms we developed to enable the effective analysis of ML pipelines. Finally, we briefly describe the implementation details of our system.

3.1 Domain Requirements

We conducted interviews with six data scientists who actively work with AutoML systems in the context of the D3M project [19]: the developers of four distinct AutoML systems (D1 - D4) and two data scientists that are tasked with evaluating the D3M AutoML systems (E1 and E2). Since each developer works on a specific system, they have different needs and follow distinct workflows. However, they share some challenges. The AutoML evaluators are part of the D3M management team. They are responsible for selecting what types of ML tasks the developers must focus on and also evaluate system performance.

D3M pipelines are represented as JSON-serialized objects that contain metadata, input and output information, and the pipeline architecture, which is described as a directed acyclic graph (DAG) [12, 34]. The exploration of ML pipelines collections is a task performed by all AutoML developers and evaluators. All interviewees said they explored pipelines by looking at their JSON representations, and complained that reading the text files, and inspecting the pipelines one at a time was a tedious and time-consuming task. Understanding and comparing the pipelines is difficult, in particular, since the DAG structure is hard to grasp from the JSON representaion.

D1 said she does not have time to inspect pipelines often, and instead focuses on assessing cross-validation scores and looking for correla-

tions between primitives and performance scores. In contrast, D2, D3 and D4 they examine the pipeline DAGs and their architecture. D1 and D2 also analyze the prediction and training time. They mentioned that their AutoML systems were evaluated within a given time budget, therefore training time is an important metric for them.

D2's system has a blacklisting feature: when a primitive is found to have poor performance, it can be flagged and excluded from the search process. Therefore, he was interested in identifying when a primitive was associated with high and l scores for pipelines.

D3 usually compares pipelines using their cross-validation scores. When he finds a problem for which his systems derives sub-optimal pipelines, he inspects the pipelines derived by other systems. His goal is to understand which features in his pipelines lead to the low scores, and conversely, why the pipelines derived by the other systems perform better. By answering these questions, he hopes to gain insights into if and how he can improve his system. He is also interested in exploring the pipelines at the hyperparameter level, but said this is currently not possible due to the large number of primitives (over 300), pipelines, and parameters involved.

D4 is also interested in comparing pipelines, albeit for a different reason. More specifically, he is interested in comparing AutoML pipelines from different sources, including human generated pipelines. His goal is to evaluate if there are differences between machine- and human-generated pipelines. He is also interested in primitive similarity. More specifically, he wants to find which primitives are exchangeable within a pipeline architecture.

The analysis workflow followed by the AutoML evaluators is significantly different from that of the developers. While developers focus on pipeline structure, evaluators are mostly concerned with how well the systems perform and the problem types (e.g., classification, regression, forecasting, object detection, etc.) they currently support and should support in future iterations. More specifically, E1 and E2 said that their workflow consisted mostly on evaluating AutoML systems based on their cross-validation scores. However, they were also interested in checking how the primitives were being used, and whether AutoML systems produced different pipelines a given problem type. More specifically, they stated that if all AutoML systems derived the same (or very similar) pipelines, the task they are solving is no longer challenging and new problem types should be proposed. For formal evaluations, the D3M systems are evaluated using sequestered problems that are not visible to the developers. Thus, to give actionable insight to AutoML developers without disclosing specifics of the sequestered problems, E2 was also interested in identifying why pipelines fail.

We compiled the following desiderata from the interviews:

[R1] Pipeline collection overview and summary: all participants would like to visualize and compare multiple pipelines simultaneously, instead of inspecting them one by one.

[R2] Primitive usage: E1 and E2 are interested in exploring how primitives are used across different AutoML systems. More specifically, they want to check if the systems are generating diverse solutions and if there are underutilized primitives.

[R3] Visualizing primitive hyperparameters: D3 would like to be able to explore the hyperparameter space of the primitives used in his pipelines.

[R4] Visualizing pipeline metadata: D1, D2, E1 and E2 mentioned they were interested in visualizing and comparing different aspects of the trained pipelines, including scores, prediction and training time.

[R5] Finding correlations between primitives and scores: D1 and D2 were interested in identifying primitives that correlate with high scores on different problems and datasets. Furthermore, D2 would like to see primitives that perform poorly in order to blacklist them, and E2 is interested in identifying possible causes for pipeline failure (i.e., low scores).

[R6] Visualizing and comparing pipeline graphs: all developers were interested in visualizing the connections between pipelines primitives using a graph metaphor. Furthermore, D3 and D4 are interested in performing a detailed comparison of the pipeline graphs. In particular, they want to identify how different AutoML systems structure their pipelines to solve a particular problem type.

3.2 Visualization Design

In order to fulfill the requirements identified in the previous section, we developed *PipelineProfiler*, a tool that enables the interactive exploration of pipelines generated by AutoML systems. Fig. 1 shows *PipelineProfiler* being applied to compare pipelines derived by three distinct AutoML systems for a classification problem that aims to predict the presence of heart disease using the *Statlog (Heart) Data Set* [18]. The main components of *PipelineProfiler* are the Pipeline Matrix (C) and the Pipeline Comparison View (D). The Pipeline Matrix (C) shows a tabular summary of all the pipelines in the collection. The user can also drill down and explore one or multiple pipelines in more detail – the graph structure of selected pipelines are displayed in the Pipeline Comparison View (D) upon request. The system Menu (B) enables users to focus on a subset of the pipelines, export pipelines of interest to Python, sort the table rows and columns, and perform automated analyses over groups of primitives. These operations are described later in this section. *PipelineProfiler* is implemented as a Python library that can be used with Jupyter Notebooks to facilitate the integration with the workflow of the AutoML community (A).

Pipeline Matrix

The Pipeline Matrix provides a summary for a collection of machine learning pipelines [**R1**] selected by the user. Its visual encoding was inspired by visualizations used for topic modeling systems [1, 9]. However, instead of words and documents, this matrix represents whether a primitive is used [**R2**] in a machine learning pipeline (Fig. 1(C1)). Users can interactively reorder rows and columns according to pipeline evaluation score, pipeline source (AutoML system that generated it), primitive type (e.g., classification, regression, feature extraction, etc.), and estimated primitive contribution (i.e., correlation of primitive usage with pipeline scores). Furthermore, we use shape to encode primitive types. For example, pre-processing primitives are represented by a circle, while a plus sign is used to represent feature extraction primitives (see the legend in the top-right corner of (Fig. 1).

To support the exploration of hyperparameters [**R3**], *PipelineProfiler* implements two interactions that show this information on demand: parameter tooltip and one-hot-encoded parameter matrix. When the user hovers over a cell in the matrix, a tooltip shows the primitive metadata (type and Python path) as well as a table with all the hyperparameters set. Fig. 1(C2) shows a tooltip for primitive *Denormalize*, with four hyperparameter values set. Users can also inspect a summary of hyperparameter space for a primitive by selecting a column in the Pipeline Matrix. When a primitive (column) is selected, all of its hyperparameters are represented using a one-hot-encoding approach: each hyperparameter value becomes a column in the matrix, and dots indicate when the hyperparameter is set in a pipeline. Fig. 1(C3) shows the hyperparameter space of *Xgboost Gbtree*.

Domain experts were interested in exploring pipeline metadata [**R4**], including training and testing scores, training time and execution time. *PipelineProfiler* shows the pipeline metadata in the Metric View (Fig. 1(C4)). Users can select which metric to display using a drop down menu, and the numerical values are shown in bar chart aligned with the matrix rows. In C4, the user can choose to display the metric *F1* or the prediction time. Pipeline rows can be re-ordered based on the metric, and to enable a comparison across systems, users can also interactively group pipelines based by the system that generated them.

To convey information about the relationships between primitive usage and pipeline scores [**R5**], we designed the Primitive Contribution view. This view shows an estimate of how much a primitive contributes to the score of the pipeline using a bar chart encoding, aligned with the columns of the matrix (Fig. 1(C5)). The contribution can be either positive or negative, representing positive or negative primitive correlation with the scores. For example, in C5, *Deep Feature Synthesis* is the primitive most highly correlated with *F1*.

(a) Best performing pipeline ■ (F1 Macro: 0.45)

(b) Worst performing pipeline ■ (F1 Macro: 0.06)
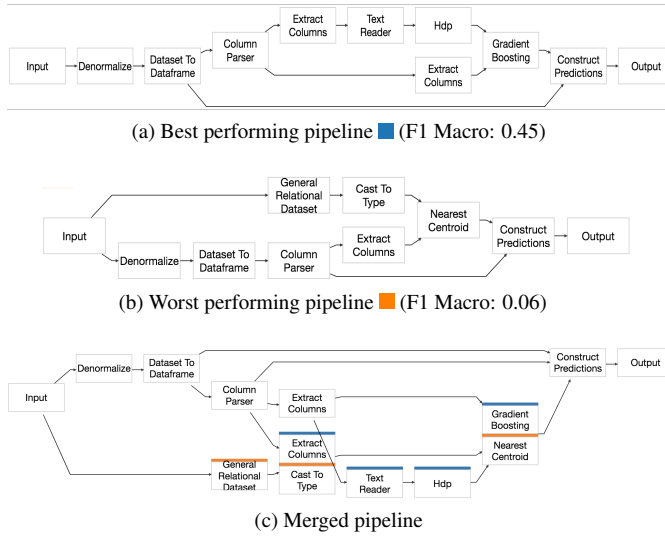

(c) Merged pipeline

Fig. 2. Pipeline Comparison View, showing the best and worst pipelines for a multitask classification problem on the *20 newsgroups* dataset. (a) and (b) show individual pipeline structure for the best and worst pipelines respectively. (c) presents the merged view of both pipelines, highlighting the differences between them using color-coded headers. In the merged pipeline, blue headers represent primitives that only appear in (a) and orange headers, primitives that only appear in (b). Primitives without a color-coded header are shared by both pipelines.

We estimate the primitive contribution using the Pearson correlation between the primitive indicator vector p ($p_i = 1$ if pipeline i contains the primitive in question and $p_i = 0$ otherwise) and the pipeline metric vector m, where m_i is the metric score for pipeline i. Since p is dichotomous and m is quantitative, the Pearson correlation can be computed more efficiently with the Point-Biserial Correlation (PBC) coefficient [43]. PBC is equivalent to the Pearson correlation, but can be evaluated with fewer operations. Let $\overline{m_1}$ be the mean of the metric score (m) when the primitive is used ($p_i = 1$); $\overline{m_0}$, the mean of the scores when the primitive is not used ($p_i = 0$); s be the standard deviation of all the scores (m); n_1 be the number of pipelines where the primitive is used; n_0 be the number of pipelines where the primitive is not used; and $n = n_1 + n_2$. The point-biserial correlation is computed as: $PBC = \left[\frac{\overline{m_1} - \overline{m_0}}{s}\right]\sqrt{\frac{n_1 n_0}{n^2}}$

Pipeline Comparison View

To provide a concise summary of a collection of pipelines, the Pipeline Matrix models the pipelines as a set of primitives that can be effectively displayed in a matrix. However, while analyzing pipelines collections, AutoML developers also need to examine and compare the graph structure of the pipelines [**R6**]. The Pipeline Comparison view (Fig. 1(D)) consists of a node-link diagram that shows either an individual pipeline, or visual-difference summary of multiple pipelines selected in the matrix representation. In the summary graph, each primitive (node) is color-coded to indicate the pipeline where it appears. If a primitive is present in multiple pipelines, all corresponding colors are displayed. If a primitive appears in all selected pipelines, no color is displayed. The Pipeline Comparison View enables users to quickly identify similarities and differences across pipelines. Fig. 2 shows the best (a) and worst pipelines (b) solving the *20 newsgroups classification problem* [32], and a merged pipeline (c) that highlights the differences between the two pipelines, clearly showing that the best pipeline (blue) uses a Gradient Boosting classifier, and an HDP and Text Reader feature extractors.

To support the comparison of multiple pipeline structures, we adapted the graph summarization method proposed by Koop et al. [30]. Since ML pipelines are directed-acyclic graphs, we modify the method to avoid cycles in the merged graph. The algorithm creates a summary graph by iteratively merging graph pairs. The merge of two graphs G_1 and G_2 is performed in four steps, as detailed below.

1) Computing Node Similarity: Let $p \in G_1$ and $q \in G_2$ be two primitives (nodes). We say that p and q have the same type if they perform the same data manipulation (e.g., Classification, Regression, Feature Extraction, etc.). The similarity f is given by:

$$f(p,q) = \begin{cases} 1.0 & \text{if } p \text{ and } q \text{ are the same primitive (total match)} \\ 0.5 & \text{if } p \text{ and } q \text{ have the same type (partial match)} \\ 0.0 & \text{otherwise} \end{cases}$$

As in Koop et al. [30], we use the Similarity Flooding algorithm [33] to iteratively adjust the similarity between nodes and take node connectivity into account. We refer the reader to [33] for details.

2) Graph Edit Matrix Construction: In order to match two graphs, $G1$ and $G2$, the algorithm builds a graph edit matrix that contains the all the possible costs to transform $G1$ into $G2$. Let m and n be the number of nodes G_1 and G_2 respectively. The edit matrix E is defined so that the selection of one entry from every row and one entry from every column corresponds to a graph edit that transforms G_1 into G_2 [40]. E contains the costs to add (a), delete (d) and substitute (s) nodes. We choose costs that prioritize node substitutions in case of a total or partial match: $s_{i,j} = 1 - f(i, j)$, $d = 0.4$ and $a = 0.4$.

$$E = \begin{bmatrix} s_{00} & s_{01} & \cdots & s_{0n} & a_0 & \infty & \cdots & \infty \\ s_{10} & s_{11} & \cdots & s_{1n} & \infty & a_1 & \cdots & \infty \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ s_{m0} & s_{m1} & \cdots & s_{mn} & \infty & \infty & \cdots & a_m \\ d_0 & \infty & \cdots & \infty & 0 & 0 & 0 & 0 \\ \infty & d_1 & \cdots & \infty & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \infty & \infty & \cdots & d_n & 0 & 0 & 0 & 0 \end{bmatrix}$$

3) Node matching: We use the Hungarian algorithm [31] to select one entry of every row and one entry of every column of E, while minimizing the total cost of the graph edit. Two nodes match when one can be substituted by the other, i.e., their substitution entry is selected from the matrix.

4) Graph merging: We merge G_1 and G_2 by creating a compound node for every pair of nodes that were matched in step 3. However, since machine learning pipelines are directed acyclic graphs, we do not want the merged graph to have cycles either. Therefore, we use the additional constraint to only merge nodes that do not result in cycles in the merged graph. This check is done using a depth search first after each merge.

Combined-Primitive Contribution

The primitive contribution presented in the previous section does not take into account primitive interactions. For example, it might be the case that for a given problem, the classification algorithm *SVM* and the pre-processing *PCA* together produce good models , but they may lead to low-scoring pipelines when used independently. Because the contribution is estimated with the Point-Biserial Correlation of the binary primitive usage vector and pipeline score, interactions involving multiple primitives are not considered.

To take all primitive interactions into consideration, it would be necessary to check for the correlations of all the primitive groups in the powerset of our primitive space. This strategy has two critical problems: 1) it is not computationally tractable, and 2) it would result in a number of combinations prohibitively large for users to inspect. To tackle this challenge, we propose a new algorithm to identify groups of primitives strongly correlated with pipeline scores. The algorithm works as follows: for every combination of primitives S up to a predefined constant size, 1) create a new primitive indicator vector $\hat{p} = \prod_{i \in S} p_i$,

which contains 1 if the set of primitives is used in the pipeline, and 0 otherwise. 2) compute the correlation of the primitive group with the pipeline scores using the Point-Biserial Correlation (comparing the pipelines that have vs do not have the combination of primitives). 3) select which combination of primitives to report to the user. We only

report the primitive group if its Pearson correlation is greater than the Pearson correlation of all the elements in its powerset. Algorithm 1 describes CPC in detail.

Algorithm 1 Combined-Primitive Contribution

Require: $p_1, p_2, \ldots, p_n$, the primitive indicator vectors
Require: m, the evaluation metric score vector
Require: K, the maximal cardinality of the primitive group
  $I \leftarrow [1, 2, \ldots, n]$
  $contributions \leftarrow \{\}$
  //Computing correlations for all groups of primitives up to size K
  //$2^I_{\leq K}$ is the powerset of I up to cardinality K
  for $S \in 2^I_{\leq K}$ **do**
    $\hat{p} \leftarrow \prod_{i \in S} p_i$
    $contributions[S] \leftarrow corr(\hat{p}, m)$
  end for

  //Selecting the combination of primitives to report to the user (R)
  $R \leftarrow [\,]$
  //$2^I_{\geq 2, \leq K}$ is the powerset of I of cardinality c: $2 \leq c \leq K$
  for $S \in 2^I_{\geq 2, \leq K}$ **do**
    $keep \leftarrow$ **true**
    //Checks if there is a subset of S with greater contribution
    for $sub \in 2^S_{\geq 1}$ **do**
      $a \leftarrow contributions[S]$
      $b \leftarrow contributions[sub]$
      if $|b| \geq |a|$ **then**
        $keep \leftarrow$ **false**
        break
      end if
    end for
    if $keep =$ **true then**
      $R = R \cup S$
    end if
  end for
  return R

The idea behind CPC is simple. The algorithm checks the correlation between combinations of primitives and the pipeline scores, and reports surprising combinations to the user (correlations not shown in the Primitive Contribution View). The user defines K (in our tests, we found that $K = 3$ is effective). If there are n primitives, the algorithm evaluates $\binom{n}{K}$ groups of primitives and has a time complexity of $O\left(\binom{n}{K}\right)$. In *PipelineProfiler*, this CPC can be run via the "Combinatorial Analysis" menu (Fig. 1(A)). When the algorithm is run, we show a table containing the selected groups of primitives and the correlation values. Fig. 3 shows an example of a CPC run over pipelines derived to perform a classification task using the Diabetes dataset [18].

Implementation details

PipelineProfiler is implemented as a Python 3 library. The front-end is implemented in Javascript with React [20], D3 [4] and Dagre [10]. The back-end, responsible for data management, graph merging and the Jupyter Notebook hooks is implemented in Python with Numpy [51] and Network [27].

The *PipelineProfiler* library takes as input a Python array of pipelines in the D3M JSON [47] format, and plots the visualization in Jupyter using Jupyter Widgets hooks. We implemented a bi-directional communication between Jupyter Notebook and our tool. From Jupyter, the user can create an instance of *PipelineProfiler* for their dataset of choice. The main menu (Fig. 1(B)) of *PipelineProfiler*, on the other hand, enables users to subset the data (remove pipelines from the analysis), reorder pipelines according to different metrics, and export the selected pipelines back to Python. The goal of our design is to provide a seamless integration with the existing AutoML ecosystem and Python libraries, and to make it easier for experts to explore, subset, combine and compare results from multiple AutoML systems.

Fig. 3. Combined-Primitive Contribution applied to pipelines that solve a classification problem on the Diabetes [18] dataset: A) The Pipeline Matrix representation of the pipeline collection. B) The Combinatorial Analysis View, showing a group of two primitives, *Min Max Scaler* and *RBF Sampler*, that correlate with higher F1 scores. The two primitives are highlighted in (A). Notice that pipelines with higher scores use both primitives (#1, #2) – pipelines that use them separately have lower scores (#3 - #9).

PipelineProfiler is already being used in production by the DARPA D3M project members. An open-source release is available at `https://github.com/VIDA-NYU/PipelineVis`.

4 EVALUATION

To demonstrate the usefulness of *PipelineProfiler*, we present case studies that use a collection containing 10,131 pipelines created as part of the D3M programs Summer 2019 Evaluation. In this evaluation, 20 AutoML systems were run to solve various ML tasks (classification, regression, forecasting, graph matching, link prediction, object detection, etc.) over 40 datasets, which covered multiple data types (tabular, image, audio, time-series and graph). Each AutoML system was executed for one hour and derived zero or more pipelines for each each dataset.

4.1 Case Study 1: Improving an AutoML System

To showcase how *PipelineProfiler* supports the effective exploration of AutoML-derived pipelines in a real-world scenario, we describe how an AlphaD3M developer used the system, the insights he obtained, and how these insights helped him improve AlphaD3M. AlphaD3M is an AutoML system based on reinforcement learning that uses a grammar (set of primitives patterns) to reduce the search space of pipelines [16, 17].

The AlphaD3M developer started his exploration using a problem for which AlphaD3M had a poor performance: a multi-class classification task using the *libras move* dataset[1] from the OpenML database [50]. For this task, in the ranking of all pipelines produced by D3M AutoML systems, the best pipeline produced by AlphaD3M was ranked 18th with an accuracy score of 0.79.

Comparing pipeline patterns. The developer sought to identify common patterns in the best pipelines that were overlooked by the AlphaD3M search. To this end, he first sorted the primitives by type and the pipelines by performance. This uncovered useful patterns. As Fig. 4A shows, primitives for feature selection were frequently present in the best pipelines, while lower-scoring pipelines did not use these primitives. Although he identified other patterns, the information provided by primitive contribution bar charts indicated that feature selection primitives had a large impact in the score of best pipelines. This information led the developer to hypothesize that the usage of feature selection primitives might be necessary for pipelines to perform well for the problem and data combination.
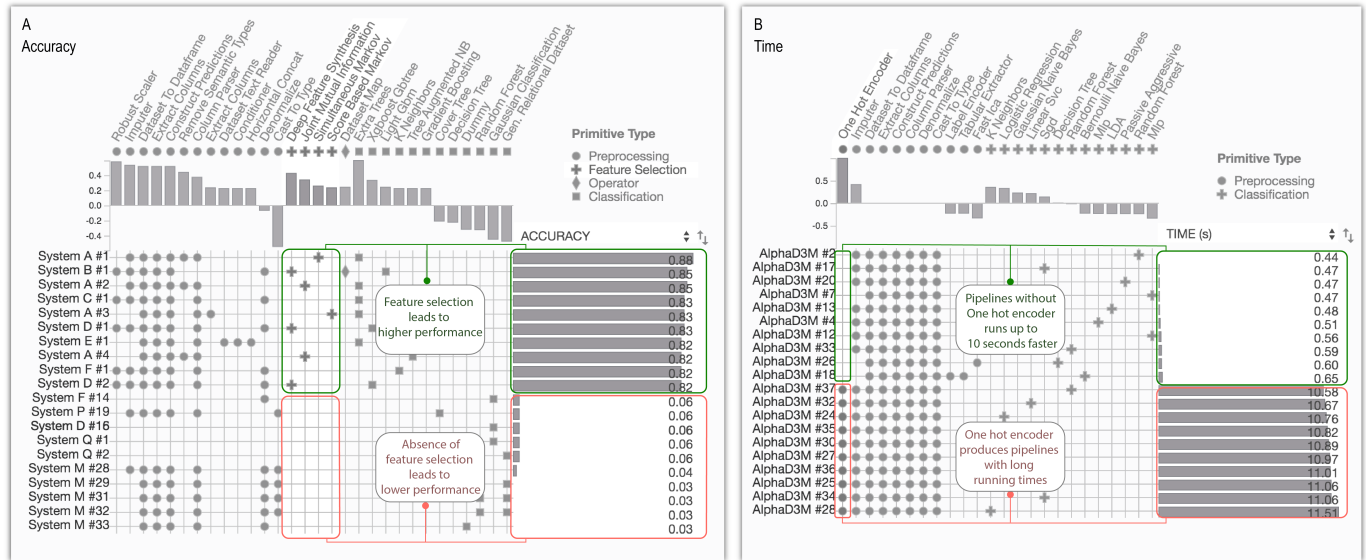
---

[1] OpenML dataset, https://www.openml.org/d/299

Fig. 4. Pipelines Matrix. A) Pipelines are sorted by performance, the 10 best pipelines at the top and the 10 worst pipelines at the bottom. Green and red boxes show the presence and absence, respectively, of feature selection primitives in the pipelines with their performances. B) Pipelines are sorted by execution time, only pipelines generated by *AlphaD3M* are displayed. Green and red boxes show the absence and presence, respectively, of one-hot encoder primitive in the pipelines with their execution times.

Exploring execution times. The developer then analyzed the pipelines produced only by AlphaD3M. Fig. 4B clearly shows that pipelines containing one-hot encoding primitives take a substantially longer time to execute, approximately 10 seconds – this is in contrast to pipelines that do not use this primitive and take less than 1 second. He also saw in primitive contribution bar charts that the one-hot encoding primitive has the highest impact on the running time. Using this information, he realized that for this specific dataset, one hot encoding primitives were used inefficiently since all the features of the dataset were numeric. Since an AutoML system needs to evaluate a potentially large number of pipelines during its search, an order-of-magnitude difference in execution time, such as what was observed here, will greatly limit its ability to find good pipelines given a limited time budget – for the summer evaluation, this budget was 1 hour.

Reducing the search space. AutoML systems have to deal with large search spaces. To synthesize pipelines, AlphaD3M takes into account over 300 primitives. This often means that there is a delay for the system to derive *good* pipelines. An effective strategy to reduce the search space is the prioritization of primitives. In Fig. 5, we can see the results of the Combined-Primitive Contribution view, which shows that the combination of the primitives *Joint Mutual Information*, *Extra Trees* and *Imputer* produce good results. Using this information, the expert realized that this sequence of primitives could be added to AlphaD3Ms grammar as static components in order to reduce the search space, and consequently, to produce good pipelines faster.

Using insights to improve AlphaD3M. After the analysis, the developer modified the AlphaD3M system's handling of feature selection, the use one-hot encoding primitive, and the prioritization of primitives. Feature selection and prioritization of primitives were added to the AlphaD3M grammar and rules were added to the workflow to apply one-hot encoding primitives only for categorical features. The new version of AlphaD3M now leads the ranking for the multi-class classification task in the *libras move* dataset with an accuracy of 0.88. With respect to execution time, the current average time to evaluate each pipeline for this problem is less than 1 second, while previously it took 10 seconds. As a point of comparison, whereas the best pipeline derived by AlphaD3m after 5 minutes of search had a score of 0.74, now, the best pipeline has a score of 0.79.

4.2 Case Study 2: Exploring AutoML Approaches

AutoML systems in the D3M program use different approaches to generate pipelines. In this case study, we show the use of *PipelineProfiler*
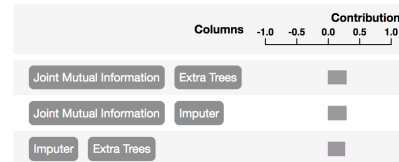

Fig. 5. Combined-Primitive Contribution: the combination of primitives *Joint Mutual Information*, *Extra Trees* and *Imputer* produce good results together.

to analyze and compare systems, and discuss some valuable insights obtained into features that impact a system's performance for a problem. An AutoML developer set out to compare how six D3M systems – denoted by *A, B,C,D, E, F* – performed for a regression task using the *cps 85 wages* dataset.[2] The systems output a total of generated 114 pipelines after 1 hour. Since *System F* produced only one pipeline, it was excluded from the comparison. *System A* obtained the best performance followed by *System B*, *System C*, *System D*, and *System E* with 20.28, 20.29, 20.68, 21.46 and 21.46 mean squared error, respectively. Using the Pipeline Comparison View, the developer could also easily see noticeable differences in the strategies used by the AutoML systems to construct the pipelines. We discuss this further below.

Template-based approaches. ML templates are manually designed to reduce the number of invalid pipelines during the search process. Although this approach reduces the search space, it also limits the exploration of potential pipelines. Fig. 6 shows the visual difference for the top-5 pipelines produced by *System D*. Note that they all have the same exact structure and only differ in the estimator used (*Ridge*, *Lars*, *Ada Boost*, *Elastic Net* or *Lasso*). A similar behavior was observed for *System E*.

Hyperparameter-tuning strategy. Analyses supported by *Pipeline-Profiler* can also provide insights into the strategies used by AutoML systems to tune hyperparameters. While exploring the pipelines produced by *System C*, the developer identified interesting patterns that uncover the strategy this system uses to tune hyperparameters. Using the Pipeline Comparison View, he noticed that some pipelines produced by this AutoML system had the same structure, but used different hyperparameter values. This is illustrated in Fig. 7A, which shows the merged graph for 4 distinct pipelines. He then inspected the hyperparameters of the *XGBoost* primitive using the one-hot-encoded parameter

---

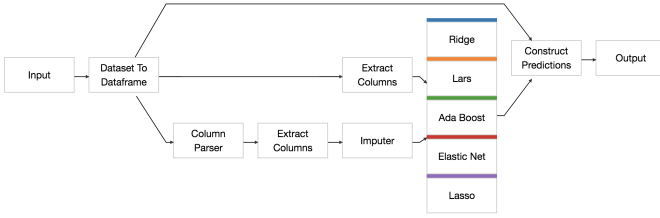[2]OpenML dataset, https://www.openml.org/d/534

Fig. 6. A visual comparison of pipelines produced by System D suggests that it fixes the pipeline structure and tries multiple regression algorithms.
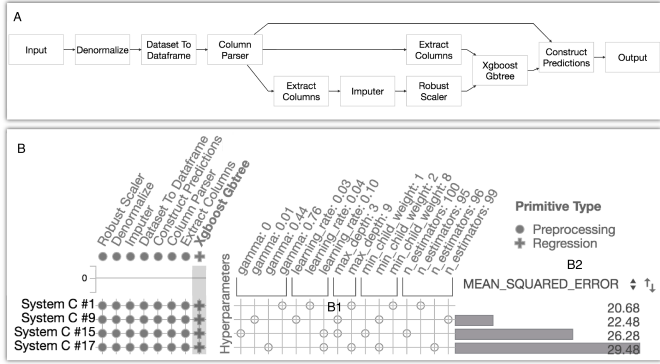

Fig. 7. *System C* produced four pipelines (#1, #9, #15 and #17) with the same graph structure, as the merged graph (A) shows. Even though these pipelines have identical structures, the hyperparameter values for the *Xgboost Gbtree* primitive are different (B1), and this results in different scores for the pipelines (B2). This pattern suggests that *System C* tunes the hyperparameter values after it derives the pipeline structure.

matrix, and observed that they had different values (see Fig. 7B). This suggests that *System C* first defines the structure of a pipeline, and then searches for the best-performing hyperparameters values. We note that the changes in these values have important impact in pipeline performance. For instance, the mean squared error for the best and worst pipeline are 20.68 and 29.48, respectively.

Search over preprocessing primitives. By exploring another set of pipelines generated by *System C* (see Fig. 8), he observed that all pipelines use the same estimator – the *XGBoost* primitive, but the preprocessing primitives differ – *Robust Scaler*, *Encoder* and *Extract Columns* are used. This suggests that *System C* also searches over alternative preprocessing sequences for a given representative ML estimator, likely in an attempt to optimize the steps for data transformation and normalization.

Full-search approach. The approach applied by *System A* and *System B* seems to search over alternative preprocessing primitives as well as estimators. Fig. 9 shows the merged graph for the top-5 pipelines derived by *System A*. Note that these pipelines differ both in the structure and primitives used. Pipelines derived *System B* display a similar behavior.

Although comparing AutoML approaches requires complex analyses, these case studies show that, by presenting an overview that highlights differences and similarities for a set of pipelines, the graph comparison view is quite effective at uncovering interesting patterns that provide insights into the search strategies employed by AutoML systems. Additional questions can be explored by drilling down into the details of the hyperparameter values.

The developer also compared the performance of the different systems. *System E* and *System D* resulted in the lowest-scoring pipelines, probably due to the unsophisticated search strategies they employ – the use of fixed templates may be pruning too much of the search space and ignoring efficient pipelines that do not follow the adopted template. On the other hand, *System A* and *System B*, which perform a broader search, created pipelines that had high scores.
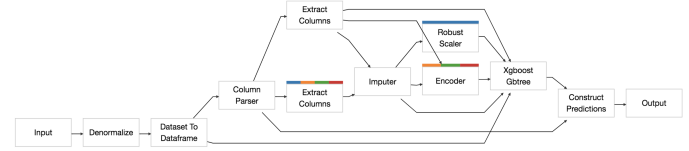

Fig. 8. A comparison of pipelines produced by *System C* indicates that, for a fixed regression algorithm (`Xgboost`), it searches for alternative sequences of preprocessing primitives.
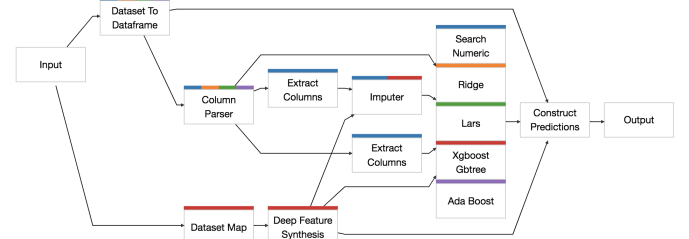

Fig. 9. A comparison of pipelines produced by `System A` shows that these pipelines vary both in structure and in the primitives used, suggesting that it performs a broad search which considers multiple preprocessing sequences and different regression algorithms.

4.3 Expert Interviews

To validate our design decisions, we conducted a second round of interview with the six data scientists from the D3M project who had previously helped us identify the system requirements (Section 3.1): four AutoML Developers (D1-D4) and two AutoML Evaluators (E1, E2). In the experiment, experts were asked to explore a dataset of their choice according to their usual pipeline exploration workflows. Developers were asked to use the system to gain insights about the AutoML strategies and identify possible modifications that could result in system improvements. Evaluators were asked to use the tool to explore the produced pipelines in order to evaluate (and compare) the AutoML systems. They were also asked how this tool could be included in their current workflows to make them more effective.

Each interview took 45 minutes and proceeded as follows. We first presented our system to the participant and clarified any questions they had (10 minutes). Then, we let them choose one problem from the D3M repository [47] to explore (30 minutes). Finally, we asked if the participant had any comments on the system (5 minutes). The problems used in this study are shown in Table 1. Note that the systems did not always produce good evaluation metrics for the chosen problems, indicating that these problems are challenging. For example, seven AutoML systems produced 115 pipelines for the *Word Levels* [26] classification problem, but no system an F-score above 0.33.

The participants were free to use *PipelineProfiler* and explore the available pipelines. They were instructed to speak while using the system, following a "think aloud" protocol. While the participants performed the task, an investigator took notes related to the actions performed. After completion, the participants filled a questionnaire to express their impressions on the usability of the system. Participants received a $20 US Dollars gift card for their participation. In this section, we describe the insights gathered by the participants.

Expert Insights

Data preprocessing. Before they started the investigation, two developers removed outliers from their datasets. D1 and D3 selected datasets with a Mean Squared Error evaluation metric, which is unbounded in the positive real numbers. The two selected datasets, Auto MPG [18] and Sunspots [44], had pipelines with error metrics above 100,000.0, which made the scales difficult to read. In both datasets, the data scientists looked at the Primitive Contribution View and noted that a problem with the *SGD* primitive was likely responsible for these high errors. They used *PipelineProfiler* subset menu to remove these pipelines from the analysis.

Table 1. Datasets used in the expert interviews

Dataset	Dataset Type	Task Type	Metric	Mean Score	Score Range	# Pipelines	# Primitives	Participant
Auto MPG [18]	Tabular	Regression	Mean Squared Error	30.18 ± 82.39	$[4.71, 595.24]$	103	71	D1
Word Levels [26]	Tabular	Classification	F1 Macro	0.24 ± 0.07	$[0.04, 0.33]$	115	69	D2
Sunspots [44]	Time Series	Forecasting	Root Mean Squared Error	27.82 ± 23.64	$[8.08, 121.27]$	137	71	D3
Popular Kids [49]	Tabular	Classification	F1 Macro	0.39 ± 0.06	$[0.21, 0.48]$	120	64	D4
Chlorine Concentration [8]	Time Series	Classification	F1 Macro	0.39 ± 0.23	$[0.00, 0.78]$	47	48	E1
GPS Trajectories [18]	Tabular	Classification	F1 Macro	0.48 ± 0.15	$[0.00, 0.68]$	163	91	E2

Performance investigation. Most participants started the analysis by looking at the performance of the pipelines. All developers were interested in how their systems compared against the others. Evaluators, on the other hand, focused on the distribution of scores across all systems. For example, the first comparison E1 did was using the pipeline scores. She grouped pipelines by source and noticed the difference in scores among the top pipelines from each AutoML system. The top two AutoML systems had pipelines with F1 Scores of 0.78 and 0.70, which she mentioned were very close. The other systems produced pipelines with much lower scores, below 0.25.

Primitive comparison. Participants were very interested in comparing the pipelines produced by different systems. In particular, developers spent a considerable amount of time comparing pipelines from their systems against pipelines from the other tools. For example, D4 inspected a classification dataset and found that while a gradient boosting algorithm was used in the top-scoring pipelines, his system was using decision trees. The Primitive Contribution view confirmed his hypothesis that the use of gradient boosting was indeed correlated with high scores. He said that he could use this insight to drop and replace primitives in his AutoML search space. D1, D2 and D3 had similar findings in their pipeline investigations. Evaluators compared primitive usage for a different reason: they wanted to make sure AutoML systems were exploring the search space and the primitives available to their systems. For example, E1 noticed that the best AutoML system used a single classifier type on its pipelines, as opposed to other systems that had more diverse solutions. E2 did a similar analysis on his dataset.

Hyperparameter search strategy. D1 noticed that the top-five pipelines belonged to the same AutoML system and were nearly identical. She explored the hyperparameters of these pipelines using the one-hot-encoded hyperparameter view, and found that although they had the same graph structure, they were using different hyperparameters for the *Gradient Boosting* primitive. She compared this strategy with another system which did not tune many hyperparameters, and concluded that tuning parameters was beneficial for this problem.

Primitive correctness. Participants also used *PipelineProfiler* to check if primitives were being used correctly. A common finding was the unnecessary use of primitives. For example, D2 found that pipelines containing *Cast to Type* resulted in lower F1 scores. He inspected the hyperparameters of this primitive and noted that string features were being converted to float values (hashes). He concluded that string hashes were bad features for this classification problem, and the *Cast to Type* primitive should be removed from those pipelines. Similar findings were obtained with *One Hot Encoder* used in datasets with no categorical features (D3, E1, E2), and *Imputer* used on datasets with no missing data (D4, E1). E2 also found incorrect hyperparameter settings, such as the use of "warm-start=true" in a *Random Forest* primitive.

Execution time investigation. D4 checked the running times of the pipelines. In particular, he was interested in verifying whether the best pipelines took longer to run. First he sorted the pipelines by score. Then, he switched the displayed metric to "Time" and noticed that, contrary to his original hypothesis, the best pipelines were also the fastest. He looked at the Primitive Contribution View in order to find what primitives were responsible for the longer running times, and identified that the *General Relational Dataset* primitive was most likely the culprit. He concluded that if he removed this primitive, he would get a faster pipeline.

Expert Feedback

We received very positive feedback from the participants. They expressed interest in using *PipelineProfiler* for their work and suggested new features to improve the system. After the think-aloud experiment, they were asked if they had any additional comments or suggestions. Here are some of their answers:

- D1 mentioned that *PipelineProfiler* is better than her current tools: "I think this is very useful, we are always trying to improve our pipelines. The pipeline scores can give you some scope, but this is doing it more comprehensively".
- D2 liked the debugging capabilities of *PipelineProfiler*: "Actually, with this tool we can infer what search strategies the AutoML is using. This tool is really nice to do reverse engineering".
- D3 particularly liked the integration with Jupyter Notebook: "I really liked this tool! It is very informative and easy to use. It works as a standalone tool without any coding, but I can make more specific/advanced queries with just a little bit of code."
- D4 wants to integrate *PipelineProfiler* into his development workflow: "The tools is great, and I as mentioned earlier, it would be even better if an API is provided to ingest the data automatically from our AutoML systems". E1 and E2 were also interested in integrating this tool with their sequestered datasets, which used for evaluation but not shared with the developers

4.4 Usability

We evaluated the usability of *PipelineProfiler* using the System Usability Score (SUS) [5], a valuable and robust tool for assessing the quality of system interfaces [3]. In order to compute the SUS, we conducted a survey at the end of the second interview: we asked participants to fill out the standard SUS survey, grading each of the 10 statements on a scale from 1 (strongly disagree) to 5 (strongly agree). The SUS grades systems on a scale between 1 and 100 and our system obtained an average score of 82.92 ± 12.37. According to Bangor et al. [3], a mean SUS score above 80 is in the fourth quartile and is acceptable.

5 CONCLUSIONS AND FUTURE WORK

We presented *PipelineProfiler*, a new tool for the exploration of pipeline collections derived by AutoML systems. *PipelineProfiler* advances the state-of-the-art in visual analytics for AutoML in two significant directions: it enables the analysis of pipelines that have complex structure and use a multitude of primitives, and it supports the comparison of multiple AutoML systems. Users can perform a wide range of analyses which can help them answer common questions that arise when they are debugging or evaluating AutoML systems. Because these analyses are scripted, they can be reproduced and re-used. We validated our system with a set of use cases that show how *PipelineProfiler* can be used to improve an open-source AutoML tool, and presented a detailed analysis of think-aloud interviews where experts reported that discovery of novel and actionable insights into their systems.

There are many avenues for future work. To increase the adoption of our tool beyond the D3M ecosystem, we plan to add support for other pipeline schemata adopted by widely used AutoML systems. On the research front, we would like to explore how to capture the knowledge derived by users of *PipelineProfiler* and use this knowledge to steer the search performed by the AutoML system, which in turn, can lead to the generation of more efficient pipelines in a shorter time. For example, if the user finds that a group of primitives work well together, they should be able to indicate this to the AutoML system, so that it can focus the search of pipelines that use these primitives.

REFERENCES

[1] E. Alexander, J. Kohlmann, R. Valenza, M. Witmore, and M. Gleicher. Serendip: Topic model-driven visual exploration of text corpora. In *Proceedings of the IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 173–182. IEEE, 2014.

[2] S. Amershi, M. Chickering, S. M. Drucker, B. Lee, P. Simard, and J. Suh. Modeltracker: Redesigning performance analysis tools for machine learning. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 337–346, 2015.

[3] A. Bangor, P. T. Kortum, and J. T. Miller. An empirical evaluation of the system usability scale. *International Journal of Human–Computer Interaction*, 24(6):574–594, 2008.

[4] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 2011.

[5] J. Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[6] D. Cashman, S. R. Humayoun, F. Heimerl, K. Park, S. Das, J. Thompson, B. Saket, A. Mosca, J. T. Stasko, A. Endert, M. Gleicher, and R. Chang. Visual analytics for automated model discovery. *CoRR*, 2018.

[7] D. Cashman, A. Perer, R. Chang, and H. Strobelt. Ablate, Variate, and Contemplate: Visual Analytics for Discovering Neural Architectures. *arXiv:1908.00387 [cs]*, July 2019. arXiv: 1908.00387.

[8] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista. The ucr time series classification archive, 2015.

[9] J. Chuang, C. D. Manning, and J. Heer. Termite: Visualization techniques for assessing textual topic models. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pp. 74–77, 2012.

[10] B. Cobarrubia. Dagre: Graph layout for javascript. `https://github.com/dagrejs/dagre/wiki`, 2018.

[11] D3M. Data-driven discovery of models (D3M) program index. `https://github.com/darpa-i2o/d3m-program-index`, 2020. Library Catalog: gitlab.com.

[12] D3M. datadrivendiscovery / metalearning. `https://gitlab.com/datadrivendiscovery/metalearning`, 2020.

[13] D3M. Index of open source D3M primitives. `https://gitlab.com/datadrivendiscovery/primitives`, 2020. Library Catalog: gitlab.com.

[14] S. Das, D. Cashman, R. Chang, and A. Endert. BEAMES: Interactive Multimodel Steering, Selection, and Inspection for Regression Tasks. *IEEE Computer Graphics and Applications*, 39(5):20–32, 2019.

[15] D. Dingen, M. V. Veer, P. Houthuizen, E. H. J. Mestrom, E. H. H. M. Korsten, A. R. A. Bouwman, and J. V. Wijk. RegressionExplorer: Interactive Exploration of Logistic Regression Models with Subgroup Analysis. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2018.

[16] I. Drori, Y. Krishnamurthy, R. Rampin, R. d. P. Lourenco, J. P. Ono, K. Cho, C. Silva, and J. Freire. AlphaD3M: Machine Learning Pipeline Synthesis. In *Proceedings of the ICML Workshop on Automatic Machine Learning*, p. 8, 2018.

[17] I. Drori, Y. Krishnamurthy, R. Rampin, R. d. P. Lourenco, J. P. Ono, K. Cho, C. Silva, and J. Freire. Automatic machine learning by pipeline synthesis using model-based reinforcement learning and a grammar. In *Proceedings of the ICML Workshop on Automatic Machine Learning*, 2019.

[18] D. Dua and C. Graff. UCI machine learning repository, 2017.

[19] J. Elliott. Data-Driven Discovery of Models (D3M), 2020.

[20] A. Fedosejev. *React. js essentials*. Packt Publishing Ltd, 2015.

[21] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In *Advances in neural information processing systems*, pp. 2962–2970, 2015.

[22] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter. Auto-sklearn: Efficient and Robust Automated Machine Learning. In F. Hutter, L. Kotthoff, and J. Vanschoren, eds., *Automated Machine Learning: Methods, Systems, Challenges*, pp. 113–134. Springer International Publishing, Cham, 2019.

[23] Y. Gil, J. Honaker, S. Gupta, Y. Ma, V. D'Orazio, D. Garijo, S. Gadewar, Q. Yang, and N. Jahanshad. Towards human-guided machine learning. In *Proceedings of the Conference on Intelligent User Interfaces (IUI)*, pp. 614–624. ACM, 2019.

[24] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley. Google Vizier: A Service for Black-Box Optimization. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 1487–1495. Association for Computing Machinery, 2017.

[25] Google. Cloud AutoML - Custom Machine Learning Models | AutoML. `https://cloud.google.com/automl/`, 2020.

[26] O. Guzey, G. Sohsah, and M. Unal. Classification of word levels with usage frequency, expert opinions and machine learning, 2014.

[27] A. Hagberg, P. Swart, and D. S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[28] F. Hutter, L. Kotthoff, and J. Vanschoren. *Automated machine learning methods, systems, challenges*. Springer, Cham, Switzerland, 2019. OCLC: 1127437059.

[29] IBM. Watson Studio - AutoAI, Jan. 2020.

[30] D. Koop, J. Freire, and C. T. Silva. Visual summaries for graph collections. In *2013 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 57–64. IEEE, Sydney, Australia, Feb. 2013. doi: 10.1109/PacificVis.2013.6596128

[31] H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[32] K. Lang. Newsweeder: Learning to filter netnews. In *Machine Learning Proceedings 1995*, pp. 331–339. Elsevier, 1995.

[33] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the International Conference on Data Engineering*, pp. 117–128. IEEE, 2002.

[34] M. Milutinovic. *Towards Automatic Machine Learning Pipeline Design*. PhD thesis, EECS Department, University of California, Berkeley, Aug 2019.

[35] MLJAR. Machine Learning for Humans. `https://mljar.com/`, 2020.

[36] T. Mhlbacher, L. Linhardt, T. Mller, and H. Piringer. TreePOD: Sensitivity-Aware Selection of Pareto-Optimal Decision Trees. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):174–183, 2018.

[37] R. S. Olson and J. H. Moore. TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning. In F. Hutter, L. Kotthoff, and J. Vanschoren, eds., *Automated Machine Learning: Methods, Systems, Challenges*, pp. 151–160. Springer International Publishing, Cham, 2019.

[38] H. Park, J. Kim, M. Kim, J.-H. Kim, J. Choo, J.-W. Ha, and N. Sung. VisualHyperTuner: Visual Analytics for User-Driven Hyperparamter Tuning of Deep Neural Networks. *Proceedings of the 2nd SysML Conference*, p. 2, 2019.

[39] D. Ren, S. Amershi, B. Lee, J. Suh, and J. D. Williams. Squares: Supporting Interactive Performance Analysis for Multiclass Classifiers. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):61–70, Jan. 2017. doi: 10.1109/TVCG.2016.2598828

[40] K. Riesen and H. Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision computing*, 27(7):950–959, 2009.

[41] A. Santos, S. Castelo, C. Felix, J. P. Ono, B. Yu, S. R. Hong, C. T. Silva, E. Bertini, and J. Freire. Visus: An Interactive System for Automatic Machine Learning Model Building and Curation. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics (HILDA)*, pp. 1–7. Association for Computing Machinery, 2019.

[42] Z. Shang, E. Zgraggen, B. Buratti, F. Kossmann, P. Eichmann, Y. Chung, C. Binnig, E. Upfal, and T. Kraska. Democratizing data science through interactive curation of ml pipelines. In *Proceedings of the Conference on Management of Data (SIGMOD)*, p. 11711188. Association for Computing Machinery, 2019.

[43] D. J. Sheskin. *Handbook of parametric and nonparametric statistical procedures*. crc Press, 2003.

[44] SILSO World Data Center. The international sunspot number. *International Sunspot Number Monthly Bulletin and online catalogue*, 2019.

[45] T. Swearingen, W. Drevo, B. Cyphers, A. Cuesta-Infante, A. Ross, and K. Veeramachaneni. Atm: A distributed, collaborative, scalable system for automated machine learning. In *2017 IEEE International Conference on Big Data (Big Data)*, pp. 151–162, Dec 2017. doi: 10.1109/BigData.2017.8257923

[46] J. Talbot, B. Lee, A. Kapoor, and D. S. Tan. EnsembleMatrix: interactive visualization to support machine learning with multiple classifiers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1283–1292. ACM, 2009.

[47] D. M. Team. D3m metalearning database. `https://metalearning.datadrivendiscovery.org/`, 2020.

[48] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification

algorithms. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 847–855, 2013.

[49] J. Vanschoren. OpenML- PopularKids, 2014. Library Catalog: www.openml.org.

[50] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013. doi: 10.1145/2641190.2641198

[51] S. v. d. Walt, S. C. Colbert, and G. Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[52] D. Wang, J. D. Weisz, M. Muller, P. Ram, W. Geyer, C. Dugan, Y. Tausczik, H. Samulowitz, and A. Gray. Human-ai collaboration in data science: Exploring data scientists' perceptions of automated ai. *Proceedings of the ACM on Human-Computer Interaction*, 3(CSCW):1–24, 2019.

[53] Q. Wang, Y. Ming, Z. Jin, Q. Shen, D. Liu, M. J. Smith, K. Veeramacha-neni, and H. Qu. ATMSeer: Increasing Transparency and Controllability in Automated Machine Learning. *arXiv:1902.05009 [cs, stat]*, Feb. 2019. arXiv: 1902.05009. doi: 10.1145/3290605.3300911

[54] Q. Wang, J. Yuan, S. Chen, H. Su, H. Qu, and S. Liu. Visual Genealogy of Deep Neural Networks. *IEEE Transactions on Visualization and Computer Graphics*, p. 12, 2019.

[55] D. K. I. Weidele. Conditional Parallel Coordinates. In *2019 IEEE Visualization Conference (VIS)*, pp. 221–225, Oct. 2019. ISSN: null. doi: 10.1109/VISUAL.2019.8933632

[56] D. K. I. Weidele, J. D. Weisz, E. Oduor, M. Muller, J. Andres, A. Gray, and D. Wang. AutoAIViz: Opening the Blackbox of Automated Artificial Intelligence with Conditional Parallel Coordinates. *arXiv:1912.06723 [cs, stat]*, Jan. 2020. arXiv: 1912.06723. doi: 10.1145/3377325.3377538