



AFRL-RI-RS-TR-2024-009

STREAMING MODEL DESIGN, COMPARISON AND CURATION

NEW YORK UNIVERSITY

JANUARY 2024

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

AFRL-RI-RS-TR-2024-009 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /

EDWARD A. VERENICH
Work Unit Manager

/ S /

JULIE BRICHACEK
Chief, Information Systems Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

1. REPORT DATE		2. REPORT TYPE		3. DATES COVERED	
JANUARY 2024		FINAL TECHNICAL REPORT		START DATE MARCH 2017	END DATE DECEMBER 2023
4. TITLE AND SUBTITLE STREAMLINING MODEL DESIGN, COMPARISON AND CURATION					
5a. CONTRACT NUMBER FA8750-17-2-0098		5b. GRANT NUMBER N/A		5c. PROGRAM ELEMENT NUMBER 62702E	
5d. PROJECT NUMBER		5e. TASK NUMBER		5f. WORK UNIT NUMBER R27V	
6. AUTHOR(S) Juliana Freire, Roque Lopez, Aecio Santos, Claudio Silva					
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) New York University 370 Jay Street, 11th floor Brooklyn NY 11201				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/RISC 525 Brooks Road Rome NY 13441-4505			10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/RI		11. SPONSOR/MONITOR'S REPORT NUMBER(S) AFRL-RI-RS-TR-2024-009
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT The growing volumes of available digital data coupled with machine learning and simulation models have substantially increased our ability to derive insights and predictions. However, there are important obstacles to the widespread and effective use of models: data science expertise is required to create the models, and even for data scientists, it is challenging to understand, validate, and build trust on their results. The DARPA Data-Driven Discovery of Models (D3M) program had as its goal to "develop automated model discovery systems" which both empower domain experts who do not have data science training to construct empirical models that to go from raw data to accurate predictions, and to make expert data scientists more productive via automation. In our project, we have developed methods and tools that support automatic machine learning. We have addressed three key problem areas: 1) the automatic and efficient synthesis of end-to-end machine learning (ML) pipelines for a wide range of tasks; 2) enabling domain experts to specify tasks as well as explore and assess the derived pipelines; 3) support for data discovery to augment training data and improve models.					
15. SUBJECT TERMS DARPA D3M, automated machine learning, dataset discovery, model explainability, visual analytics					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT		18. NUMBER OF PAGES
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			
19a. NAME OF RESPONSIBLE PERSON EDWARD A. VERENICH				19b. PHONE NUMBER (Include area code) N/A	

Contents

List of Figures	ii
List of Tables	ii
1 SUMMARY	1
2 INTRODUCTION	2
3 METHODS, ASSUMPTIONS, AND PROCEDURES	3
3.1 AutoML	3
3.1.1 AlphaD3M	4
3.1.2 Alpha-AutoML	10
3.2 Exploring and Interacting with Models	10
3.2.1 Visus	11
3.2.2 PipelineProfiler	15
3.2.3 Understanding the Requirements for Model Interpretability	16
3.3 Data Discovery for Augmentation	17
3.3.1 The Auctus Dataset Search Engine	17
3.3.2 Auctus User Interface	19
3.3.3 Scalability and APIs	21
3.3.4 Discovering Datasets on the Web	21
3.3.5 Join-Correlation Discovery Queries	22
3.4 Results and Discussion	24
3.4.1 Auctus	24
3.4.2 Visus	29
3.4.3 AlphaD3M	31
3.4.4 Alpha-AutoML: JIATF Applications	33
4 CONCLUSIONS	34
5 REFERENCES	34

List of Figures

1	Overview of AlphaD3M.	5
2	AlphaD3M iterative improvement (left); AlphaD3M game representation (right).	5
3	(a) A code snippet to solve a semi-supervised classification task. (b) AlphaD3M allows users to inspect the contents of the input dataset, including column statistics and data types. (c) Analyzing ML pipelines through the integration with PipelineProfiler.	9
4	A code snippet to add primitives to Alpha-AutoML on the fly.	10
5	Our proposed framework encompasses problem definition, model generation, and model selection.	12
6	Visus’s data selection and problem definition screens: (A) select or load dataset view, (B) select task view (create a new problem, or load an previously created problem), (C) define problem view (displays data summaries and allows selection of problem parameters), (D) configure search view (allows setup of additional model search parameters).	13
7	Explore models view	14
8	Model explanation support for classification models: (1) rule matrix plot and (2) confusion matrix, and for regression models: (3) partial dependence plot and (4) confusion scatter plot.	15
9	Data augmentation support in Visus. (G) Dataset search results page, (G1) dataset details, and (G2) define problem screen showing data attribute summaries after data augmentation.	15
10	PipelineProfiler applied to the analysis of binary classification pipelines generated by five different AutoML systems.	16
11	Searching for datasets that mention “taxi” and contain records within the NYC area for the 2016-2021 period.	17
12	Auctus System Architecture.	18
13	Components of the Auctus user interface: keyword and filter-based search box (A); search results (B); dataset-snippets (B1); dataset summary (B2); dataset upload (C); dataset collection statistics (D).	19
14	(a) A data integration query in [A], and the augmentation interface in [B]. (b) Data Summary views. (c) Upload page.	20
15	Time spent by users to complete Question 4 and Question 5 for different queries.	27
16	Distribution of different features used on Question 5.	28
17	Distribution of different features used on Question 1.	28
18	Auctus: The search results, that match the keyword ‘protest’ and the grid dataset, are displayed. The SCAD dataset is selected and the augmentation is set up to perform a spatial-temporal join.	30
19	User-generated visualizations: (1) initial confusion scatter plot, (2) partial dependency plot for temperature from model A, (3) partial dependency plot for temperature from model B.	31
20	Performance of AutoML systems in OpenML Benchmark. X-axis shows the accuracy values (normalized by the best score), and Y-axis shows the IDs of the OpenML tasks.	33
21	The Alpha-AutoML app allows users to upload the data created using the text and image labeling tools and allows users to build machine learning pipelines using the Alpha-AutoML tool.	34
22	Examples of images classified by the best ML model generated by Alpha-AutoML	35

List of Tables

1	Tasks supported by different AutoML Systems.	4
2	Question success rates (accuracy).	27
3	Comparison of MT-AutoML systems with respect to the number of supported task types, winner pipelines, and average rank by each system.	32
4	Number of datasets by task type and number of solved datasets by each AutoML system for all task types covered by the D3M datasets.	32

1 SUMMARY

The growing volumes of available digital data coupled with machine learning and simulation models have substantially increased our ability to derive insights and predictions. However, there are important obstacles to the widespread and effective use of models: data science expertise is required to create the models, and even for data scientists, it is challenging to understand, validate, and build trust on their results.

The DARPA Data-Driven Discovery of Models (D3M) program had as its goal to “develop automated model discovery systems” [50] which both empower domain experts who do not have data science training to construct empirical models that to go from raw data to accurate predictions, and to make expert data scientists more productive via automation.

In our project, we have developed methods and tools that support automatic machine learning. We have addressed three key problem areas: 1) the automatic and efficient synthesis of end-to-end machine learning (ML) pipelines for a wide range of tasks; 2) enabling domain experts to specify tasks as well as explore and assess the derived pipelines; 3) support for data discovery to augment training data and improve models. Our key technical contributions are:

- A novel deep-reinforcement-based algorithm that learns to synthesize end-to-end ML pipelines [47, 48]. The ability to learn how to assemble these pipelines is key to support a wide range of ML tasks as envisioned by the D3M program.
- Two open-source AutoML Python libraries, AlphaD3M¹ and Alpha-AutoML², that support a wide range ML tasks [81]. According to our most recent results, AlphaD3M achieved the best performance among the other systems developed in the program. Also, it produced pipelines whose performance is superior or on par with the other state-of-the-art AutoML systems. Preliminary experiments suggest that AlphaAutoML is 20x faster than the existing, state-of-the-art D3M systems, producing high-quality models at a fraction of the time. AlphaAutoML has important benefits compared to other AutoML systems: it is extensible making it possible to integrate new developments in machine learning primitives and models; and by relying on widely-adopted open-source de-facto standards, it is easy to maintain. (see Section 3.4.3).
- Novel techniques that support human-in-the-loop interactions with the model-building process and curation of ML pipelines. We have developed two open-source systems: Visus³ [110] which enables domain experts to define problems, verify and assess the results of the automatically derived pipelines; and PipelineProfiler⁴ [96], an interactive visualization library that allows data scientists to explore and compare ML pipelines.
- A web crawler [135] and search engine for datasets⁵ [4, 22], specifically designed for data augmentation tasks in machine learning.

We have participated in different data science competitions and research events where our teams were recognized with awards, including:

- We were selected as finalists for the Wells Fargo Campus Analytics Challenge 2020⁶. Our solution was selected as one of the top 10 finalists of the challenge. We combined the capabilities of AlphaD3M and PipelineProfiler to synthesize pipelines, explore and understand ML pipelines.
- We participated in the Data Search 2 Task⁷, a shared task on ad-hoc retrieval for governmental statistical data. The official evaluation shows that our approach got the highest score among all other submitted methods.

¹AlphaD3M repository: <https://gitlab.com/ViDA-NYU/d3m/alphad3m>

²Alpha-AutoML repository: <https://github.com/ViDA-NYU/alpha-automl>

³Visus repository: <https://gitlab.com/ViDA-NYU/d3m/ta3>

⁴PipelineProfiler repository: <https://github.com/ViDA-NYU/PipelineVis>

⁵Auctus repository: <https://gitlab.com/ViDA-NYU/auctus>

⁶Wells Fargo Campus Analytics Challenge: <https://www.mindsumo.com/contests/campus-analytics-challenge-2020>

⁷NTCIR-16 Data Search 2: <https://ntcir.datasearch.jp>

- Successful participation in the Data Challenge 2023 and Halloween Hackathon 2023 organized by EOP (External Operations Platform) agency. In the latter, we obtained second-place awards in two categories “Best New Insights” and “Best User Interface Improvement”. We also got third-place awards in “Best Cross Application Integration” and “Grand Champions Winner” categories.

The technology we have developed was successfully transitioned and Alpha-AutoML has been deployed at the EOP network. The preliminary work we did on data augmentation uncovered several open research questions which we subsequently explored with funding from the National Science Foundation [61].

Besides releasing open-source tools, our research was disseminated through scientific publications. We published 19 papers at international venues and journals⁸. One of our papers received a best paper award at the ACM SIGMOD Data Management for End-to-End Machine Learning (DEEM) Workshop [134].

2 INTRODUCTION

In recent years, the growing volumes of digital data coupled with advanced machine learning techniques have created new opportunities to create models and derive data-driven predictions. However, preparing the data and creating these models require expertise in computing and data science; and even for data scientists, it is challenging to understand, validate, and build trust in their results. Consider the following scenarios.

Making NYC Traffic Safer. A group of policymakers in the New York City Mayor’s office is interested in understanding the potential impact of reducing the speed limit in NYC to 25 MPH [90]. They want to know if the number of traffic fatalities decrease; if and how this reduction will impact the traffic flow; and whether the reduced speed will significantly increase travel times to/from important destinations and between neighborhoods. To answer these questions, they need a model of the traffic in NYC.

Discovering Human Trafficking on the Web. A group of FBI agents want to discover information about human trafficking on the Web. They need to identify and monitor Web sites and pages to both detect criminal activities and build cases that can be successfully prosecuted. However, finding these sites on the Web is like searching for a needle in a haystack, thus, we need a model to automatically discover relevant content.

While building these models is a complex, time-consuming task for data scientists, it was completely out of reach for domain experts without data science training. Our goal in this project was to empower domain experts who do not have data science training to construct empirical models that go from raw data to accurate predictions, as well as to make data scientists more efficient through the automation of tasks required to create machine learning models. *We have developed novel methods and systems that, in response to experts’ questions, synthesize and refine models while at the same time enables the expert to reason about and curate the models.* We focused on three main challenges which are summarized below.

Challenge: Data Discovery. In data-driven model design, the first challenge that needs to be addressed is *data discovery*. Consider our traffic example. Because speed sensors cover only a small part of New York City (NYC), additional data is needed so that speed can be inferred for road segments without sensors. In previous work [100], we used data about taxi trips for this purpose. We developed a traffic model that relies on an efficient data-driven closest path algorithm to compute plausible routes for trips, which are subsequently used to derive accurate estimates for traffic speed and direction. This model can help answer the policy makers’ questions, but it has limitations. Since taxis do not provide complete coverage of the city, estimates for areas with a small number of taxi trips can be inaccurate. This raises a new question: are there other data sets that can complement the taxi data? In addition, several factors can affect the number of taxi trips as well as the traffic speed that must be taken into account in a model [44, 60], including holidays, special events (e.g., street closures due to parades, presidential visits), weather. Thus, a user or an automated process that creates a model should be guided to identify related and complementary data.

There are a growing number of data sets available on the Web (see e.g., [8]). However, finding relevant data is hard. Search engines are effective for searching unstructured documents, but there are currently no equivalent solutions for interrogating large collections of structured (and semi-structured) data. This problem is compounded due to the fact that data sets have incomplete schema information and are heterogeneous both syntactically and semantically.

⁸See the complete list of papers on our website: <https://vida.engineering.nyu.edu/research/d3m/publications>

We have *built a data discovery engine that assists users to identify relevant data sets across heterogeneous sources as well as expose relationships among the data sets* (Section 3.3).

Challenge: Model Synthesis. There are many challenges involved in building models. If we consider machine learning models, the common practice is for data scientists to write computer programs that encode a series of steps (or pipeline), for example, data cleaning, feature selection, the construction and parameterization of a classifier. This process often requires an iterative, trial-and-error process. For a domain expert without computer and data science training, it is hard even to get started and determine which model to try first, which parameters are important, and how these parameters should be set.

Recently, approaches for automatic machine learning (AutoML) have been proposed that aim to make machine learning more accessible to domain experts (see e.g., [95, 114]). Given a training data set, AutoML systems produce a series of models by automatically selecting learning models and parameters. They apply different methods to search over the parameter space (e.g., randomized search, bayesian optimization) and to select a model.

However, to go from raw data to predictions, complex pipelines are needed that go beyond just the creation of a learning classifier. Many other steps are needed, including data cleaning, feature selection and extraction, and as described above, interactions among features in different data sets need to be discovered. Pipeline-based systems allow users to manually construct arbitrary pipelines that encompass all these steps, but require users to have knowledge about the underlying primitives, how to configure and connect them.

We have developed a new reinforcement-learning-based method for pipeline synthesis that *automatically derives complex, end-to-end pipelines* [48]. We have also built two open-source AutoML systems – AlphaD3M and AlphaAutoML – and showed that they outperform other AutoML systems (Section 3.4.3).

Challenge: Model Exploration. Since pipeline synthesis is a best-effort approach, we need mechanisms that allow users to explore, understand, compare, and refine a potentially large number of automatically derived models. While techniques have been developed to compare models, e.g., climate models [35], learning classifiers [69, 97], they are often specific to a given domain (or model family). In addition, since we want to build pipelines that go from raw data to predictions, we need techniques that take these end-to-end pipelines into account, and not just a subset of the steps.

We have designed and implemented *interactive data visualization methods that help users understand how the different models compare in terms of accuracy, performance, features, results and decisions*.

Another important task is to *understand model input/output behavior*. Questions that we aimed to answer include: Under which circumstances do two models make different (or similar) decisions? Which decisions are correct (or incorrect) for the two models? Why did a model make a given decision? The problem of helping users understand black-box machine learning models has received substantial attention in the recent literature [3, 105]. For example, Ribeiro et al. [105] proposed LIME, a technique that aims to explain the predictions of any classifier. Their goal is to allow users to understand the reasons behind predictions, and assess whether they can be trusted. Amershi et al. [3] developed ModelTracker, an interactive visualization that supports direct error examination and debugging of machine learning models. While our design was informed by these works, we need to consider complex, end-to-end pipelines and that encode general data-driven models, not just machine learning models.

3 METHODS, ASSUMPTIONS, AND PROCEDURES

3.1 AutoML

Most of the existing AutoML systems are designed to work with tabular data and only support classification and regression problems [58, 76, 77, 94]. Cloud AutoML [66] and AutoGluon [51] also create pipelines to classify text and images and perform object detection tasks. However, these systems do not support more complex data types such as graphs, time series, audio, and video, limiting the types of problems they can address. Table 1 shows the set of task types supported by different AutoML systems.

In the context of this program, some AutoML systems have been developed to support a wide range of data types and ML tasks using an extensive set of computational primitives as building blocks – we refer to these as *multi-task* AutoML systems (MT-AutoML). MT-AutoML systems face an essential challenge: effectively searching an ample space of primitives required to synthesize pipelines for a broad range of tasks and data

Table 1: Tasks supported by different AutoML Systems.

Systems	Tabular Classification	Text classification	Image classification	Audio classification	Video classification	Tabular Regression	Clustering	Time series forecasting	Time series classification	Object detection	LUPI	Community detection	Link prediction	Graph matching	Vertex classification	Collaborative filtering	Semisupervised classification
AutoGluon	✓	✓	✓			✓		✓		✓							
AutoWEKA	✓					✓											
Auto-Sklearn	✓					✓											
Cloud AutoML	✓	✓	✓		✓	✓				✓							
H2O	✓	✓				✓											
TPOT	✓					✓											
AlphaD3M	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

types. To prune the search space, many D3M MT-AutoML systems use manually-crafted templates and grammars [33] that prescribe combinations of primitives that make sense for different problems. This, in turn, leads to other challenges: creating these templates or grammars is not only time-consuming but failing to include the necessary rules that cover the relevant primitives (and their combination) for multiple task types can negatively impact the ability of an MT-AutoML system to derive performant pipelines.

We developed AlphaD3M and Alpha-AutoML, open-source AutoML libraries that support a wide range of data and problem types. These systems combine deep reinforcement learning and meta-learning techniques to navigate the large search spaces MT-AutoML systems must navigate effectively. They include an algorithm that applies meta-learning to automatically derive task-based context-free grammars (CFGs) which cover a multitude of problems; a pipeline synthesis method based on deep reinforcement learning using sequence models with self play; and a novel search strategy that, based on previously generated pipelines and their performance, prioritizes primitives that are correlated with good pipeline performance.

AlphaD3M and Alpha-AutoML include components that aim to support usability and integration with other tasks in the data science lifecycle, from data exploration and model summarization to model deployment. It is possible to extend AlphaD3M and Alpha-AutoML, and combine them with other tools through their flexible API. For example, their integration with the PipelineProfiler [96] allows users to explore and compare the set of derived pipelines visually.

3.1.1 AlphaD3M

AlphaD3M is a *multi-task* AutoML system that supports a wide range of data and problem types (see Table 1). It is implemented in Python and can be used via *pip* installation or Docker. Figure 1 shows an overview of this library and its components. To build ML pipelines, AlphaD3M uses a rich set of primitives and a meta-learning database from the D3M ecosystem [33]. The pipeline search is conducted by four modules which: (a) automatically construct of task-specific grammars; (b) prioritize primitives that are more likely to be effective; (c) synthesize pipelines using deep reinforcement learning (Monte Carlo Tree Search and Neural Networks); and (d) tune hyperparameters. The library implements a Python API through which users can define the problem to be solved, explore the input data, obtain model summaries, analyze and compare the produced pipelines, as well as improve and deploy them.

A. The D3M Ecosystem

i) Primitives AlphaD3M uses a comprehensive collection of primitives developed by performers in the D3M program as well as from open-source libraries (e.g., scikit-learn). In total, there are 312 primitives available for different steps in ML pipelines, including data pre-processing, feature extraction, feature selection, prediction, and clustering [34], and implement state-of-the-art methods, such as ResNet50 [72], ARIMA [126], among others.

ii) The Marvin Meta-Learning Database Marvin is an open corpus of curated ML pipelines, datasets, and problems [82]. All pipelines in Marvin share the same set of primitives and are specified using the D3M

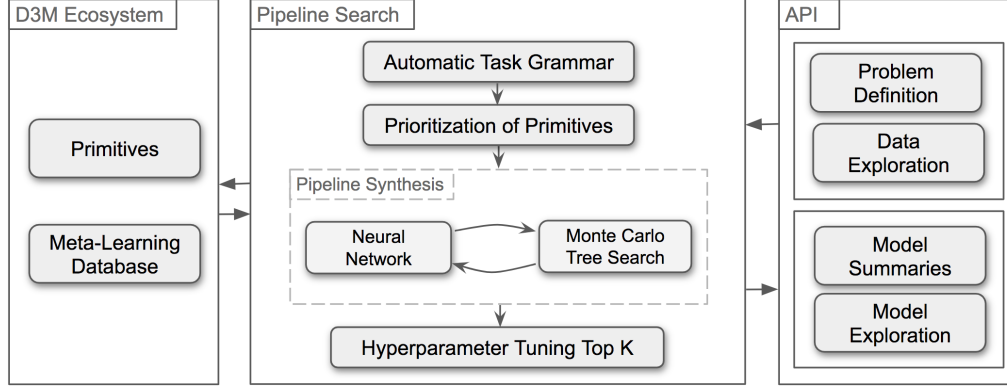


Figure 1: Overview of AlphaD3M.

format. Marvin stores approximately 2.5 million pipelines executed over 600 datasets. Since data scientists and AutoML systems that use different search strategies have produced these pipelines, the database covers a wide variety of pipeline patterns. As discussed below, we leverage the data in Marvin to assist in and improve the AlphaD3M search process. To the best of our knowledge, ours is the first work that explores this corpus.

B. Pipeline Search

The automatic synthesis of pipelines is a combinatorial problem in which we must find the best combinations of primitives and their hyperparameters. With 312 primitives and over 1,500 hyperparameters in the D3M ecosystem, the search space becomes prohibitively large. For instance, considering just the classification task over tabular data, there are 22 data cleaning, 87 data transformation, and 44 classifier primitives, leading to 84,216 possible pipelines to test. AlphaD3M uses a multi-pronged approach to manage this search space described below.

i) Pipeline Synthesis Using Deep Reinforcement Learning To synthesize the ML pipelines, AlphaD3M uses a strategy based on a single-player game technique inspired by AlphaZero [112]. It applies model-based reinforcement learning with a neural network sequence model, and a Monte Carlo Tree Search (MCTS). This section describes our representation, followed by details of the neural network and MCTS.

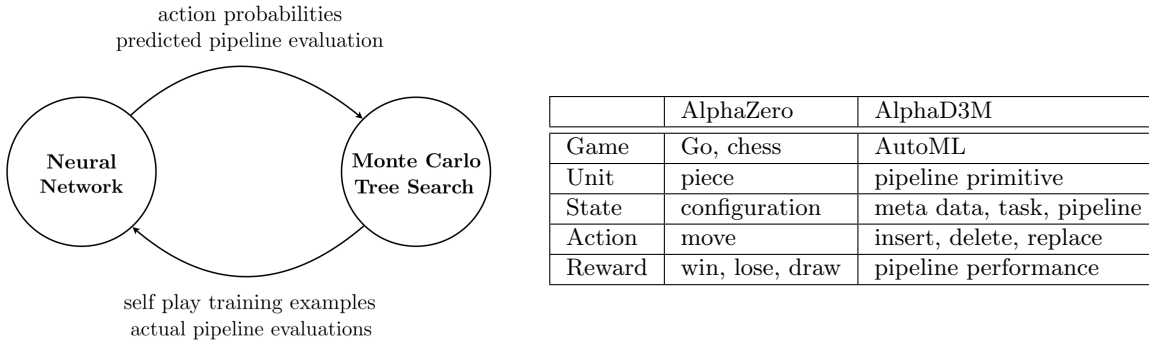


Figure 2: AlphaD3M iterative improvement (left); AlphaD3M game representation (right).

Representation Figure 2 (right) illustrates a high level analogy between a two player competitive game and our single player pipeline synthesis game, including state, action, and reward. A pipeline is a data mining work flow, of pre-processing, feature extraction, feature selection, estimation, and post-processing primitives. Algorithm 1 describes our pipeline state representation. Our architecture models meta data and an entire pipeline chain as state rather than individual primitives. A pipeline, together with the meta data and problem definition is analogous to an entire game board configuration. The actions are transitions from one state (pipeline) to another.

Algorithm 1 Pipeline Encoding

Given datasets D , tasks T , and a set of possible pipeline sequences $S_1, \dots, S_n$, from the available machine learning, and data pre and post processing primitives.

- For each dataset D_i and task T_j :
 1. Encode dataset D_i as meta data features $f(D_i)$.
 2. Encode task T_j .
 3. Encode the current pipeline at time t by a vector S_t .
 4. Encode action $f_a(S_t)$, so policy π maps $(f(D_i), T_j, S_t)$ to $f_a(S_1), \dots, f_a(S_n)$.
-

Neural Network AlphaD3M uses an LSTM. Let $f_\theta(s) = (P(s, a), v(s))$, where $P(s, a)$ is the action probabilities and $v(s)$ the evaluation score of the model predicted by the network f with parameters θ , for a given dataset D and task T , for a given state s . The neural network predicts the probabilities over actions a which lead to sequences S that describe a pipeline, which in turn solves the given task on the dataset. The network inputs are training examples (s_t, π_t, e_t) from games of self play, where s_t is the state at time t , π_t the policy estimated by MCTS, and e_t the actual pipeline evaluation at the end of the game. The state s_t is composed of a vector encoded as described in Algorithm 1. The network outputs are probabilities over actions $P(s, a)$, and an estimate of pipeline performance v .

We optimize the network parameters θ by making the predicted model S match the real world model R and the predicted evaluation results v match the real world evaluation e , by minimizing the cross entropy loss between S and R , and the mean squared error between v and e . We add an ℓ_2 regularization term for the network parameters θ to avoid over-fitting and an ℓ_1 regularization term which prefers simple pipelines. Thus our network f_θ is trained by minimizing the following non-linear loss function using stochastic gradient descent:

$$L(\theta) = S \log R + (v - e)^2 + \alpha \|\theta\|_2 + \beta \|S\|_1. \quad (1)$$

Monte Carlo Tree Search Our algorithm takes the predictions $(P(s, a), v(s))$ of the neural network and uses them in a MTCS by running multiple simulations to search for a pipeline sequence R with a better evaluation. The search result R improves upon the predicted result S given by the network by improving the network policy using the update rule:

$$U(s, a) = Q(s, a) + cP(s, a) \frac{\sqrt{N(s)}}{1 + N(s, a)}, \quad (2)$$

where $Q(s, a)$ is the expected reward for action a from state s , $N(s, a)$ is the number of times action a was taken from state s , $N(s)$ the number of times state s was visited, $P(s, a)$ is the estimate of the neural network for the probability of taking action a from state s , and c is a constant which determines the amount of exploration. At each step of the simulation, we find the action a and state s which maximize $U(s, a)$ and add the new state to the tree if it does not exist with the neural network estimates $(P(s, a), v(s))$ or call the search recursively otherwise. Next, the model represented by R is realized and applied to the data to solve the task, resulting in a better evaluation e which is the result of running the generated pipeline R on the data and task. Thus the real world search provides us with (R, e) , where R is the real world model, consisting of machine learning primitives, and e the real world evaluation of the model and pipeline using those primitives on the data and task.

The neural network predictions, the MCTS model, and the real world evaluation, together, define a loss function shown in Equation 1, which is minimized to improve the neural network parameters. This process continues iteratively until the best model, which automatically solves the task, is found.

ii) Automatic Generation of Task-Based CFG via Meta-Learning Instead of relying on manually created CFGs, we propose a new strategy that uses meta-learning to derive grammars automatically and on the fly. It does so in two steps: 1) it selects task-specific pipelines and datasets from a meta-learning database (MLDB), and 2) uses these to derive a portfolio of pipeline patterns.

Selecting Task-Oriented Datasets Since AlphaD3M supports different tasks, we need to retrieve from the Marvin MLDB pipelines produced for tasks and datasets similar to the ones we provided as inputs to the AutoML system. For instance, if we want to solve a clustering problem over a dataset D , we retrieve the pipelines used for this problem over datasets similar to D . To select relevant pipelines for a given problem P over dataset D , we use the “task keywords” tag list provided in the problem definition as features that describe the task to be solved, and search Marvin for pipelines that contain a similar set of keywords. The list is encoded as a bag-of-words (BOW). Since the set is small and most of the tags are non-standard words, e.g., *collaborativeFiltering*, *timeSeries*, it is possible to obtain accurate matches with this simple approach.

Given the set of relevant pipelines R_P , we select a subset R_{PD} containing pipelines that were applied on datasets similar to D . To determine whether two datasets are similar, we use dataset features including semantic types (e.g., categorical, date-time) and missing values, and encode them using one-hot encoding. Datasets are compared using cosine similarity.

Creating a Portfolio of Patterns After identifying similar datasets, the next step is to select the best pipelines to create a portfolio of pipeline patterns. To select these, AlphaD3M takes into consideration pipeline performance for different datasets. Some datasets are more challenging than others – the performance of a pipeline can vary widely for different datasets. To properly compare pipeline performance, AlphaD3M uses a strategy based on the *average distance to minimum* (ADTM) [127], which transforms the performance to the distance to the best-observed performance scaled between 0 and 1. In contrast to ADTM, which uses the misclassification rate, AlphaD3M uses the actual performance (the score) of the pipelines and thus, it applies the *average distance to maximum* instead to select the best pipelines. It then transforms the primitives within the pipelines to their classes. For instance, the primitive *imputer.SKlearn* belongs to the class *IMPUTATION*. If there is a pipeline with this structure: [*imputer.SKlearn svm.SKlearn*], it is converted to this pattern: [*IMPUTATION CLASSIFICATION*]. Unlike [57], which creates a unique portfolio of pipelines in an offline phase, AlphaD3M creates the portfolio online, based on the query task and dataset. Also, the output is a portfolio of patterns, not of static pipelines, which allows more flexibility to construct pipelines. These patterns are used as production rules of the grammar. Algorithm 2 describes the process of building the grammar.

Algorithm 2 Grammar Builder

Input: Marvin datasets D , query dataset q , threshold t
Initialize $S = \emptyset$ // Similar datasets
for d_i **in** D **do**
 $simTK = cosineSimilarity(getVectorTK(d_i), getVectorTK(q))$
 if $simTK > t_{sim}$ **then**
 $simST = cosineSimilarity(getVectorST(d_i), getVectorST(q))$
 if $simST > t_{sim}$ **then**
 Add d_i to S
Initialize $P = calculateADTM(S)$
Initialize $R = \emptyset$ // Production Rules
for p_i **in** P **do**
 if $performance(p_i) > t_{perf}$ **then**
 $r_i = convertToPattern(p_i)$
 Add r_i to R
return R

iii) Prioritization of Primitives In real-world scenarios, when a data scientist builds an ML pipeline, they start this process using primitives that are known to perform well. For example, XGBoost or Random Forests are good initial candidates for classification tasks. AlphaD3M follows this intuition to identify good candidate primitives for a specific task, using the data from Marvin. This prior knowledge about promising primitives can help finding better pipelines faster.

Algorithm 3 Primitives Importance

Input: Pipelines P , Patterns T
Initialize $R = \text{getPrimitives}(P)$
Initialize $G, L = []$ // Global and Local correlations
for r_i **in** R **do**
 $pc = \text{PearsonCorrelation}(r_i, P)$
 $npc = \text{normalize}(pc)$
 Add npc to G
for t_i **in** T **do**
 $p_i = \text{getPipelines}(t_i, P)$
 $R = \text{getPrimitives}(t_i, p_i)$
 for r_i **in** R **do**
 $pc = \text{PearsonCorrelation}(r_i, R)$
 $npc = \text{normalize}(pc)$
 Add npc to L
return (G, L)

Similar to [96], AlphaD3M uses Pearson Correlation (PC) to estimate how much a primitive contributes to the score of the pipeline. However, instead of using the raw scores, it uses the ADTMs values because they are scaled across different datasets. AlphaD3M estimates the primitive importance using PC 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 score vector s , where s_i is the score for pipeline i . Since p and s are dichotomous and quantitative variables, respectively, the Point-Biserial Correlation coefficient (PBC) [111] is an appropriate correlation measure – it is mathematically equivalent to the PC but can be calculated with fewer operations. The correlation values are normalized between 0 and 1 (using min-max normalization).

AlphaD3M calculates these correlations for the primitives at two levels: (a) global, when it considers all the pipelines, and (b) local, when it considers only the pipelines for each pattern. The main goal is to estimate how important a primitive is for all the pipelines and each pattern. Primitives with higher values of importance should have priority during the search of pipelines. Algorithm 3 describes the process of calculating the primitive importance values in detail. To prioritize the usage of potential primitives in AlphaD3M, it includes these values of primitive importance in the MCTS formula:

$$U(s, a) = Q(s, a) + c(\alpha P(s, a) + (1 - \alpha)R(a)) \frac{\sqrt{N(s)}}{1 + N(s, a)} \quad (3)$$

where $Q(s, a)$ is the expected reward for action a (selection of primitive a) from state s , $N(s, a)$ is the number of times action a was taken from state s , $N(s)$ is the number of times state s was visited. $P(s, a)$ are the probabilities predicted by the neural network over actions a from a state s , c is a constant which determines the amount of exploration, $R(a) = G(a) * L(a)$, $G(a)$ and $L(a)$ are the global and local importance of the action a , and α is a coefficient to keep the trade-off between $R(a)$ and $P(s, a)$.

iv) Decoupled Hyperparameter Tuning Hyperparameter tuning is an essential part of fitting machine learning models [10, 42, 115]. This is also the case for end-to-end ML pipelines that target different tasks, and all primitives contain hyperparameters, not just the estimators.

AlphaD3M performs hyperparameter tuning as an independent task, after the pipelines are constructed. It uses Bayesian optimization, which is the state-of-the-art for hyperparameter tuning [11, 42, 115] and was shown to outperform manual setting of parameters, grid search, and random search [11, 121].

AlphaD3M synthesizes and evaluates the pipelines using primitives with default values for hyperparameters. The pipelines are then ranked by performance, and the top-k pipelines are selected for tuning. AlphaD3M uses Sequential Model-Based Algorithm Configuration (SMAC) [80], a Python library for Bayesian optimization. It approximates a probability model of the performance outcome given a parameter configuration that is updated from a history of executions. AlphaD3M selects the Gaussian Processes models from SMAC to minimize an arbitrary acquisition function using the Expected Improvement criterion to choose the parameter values for

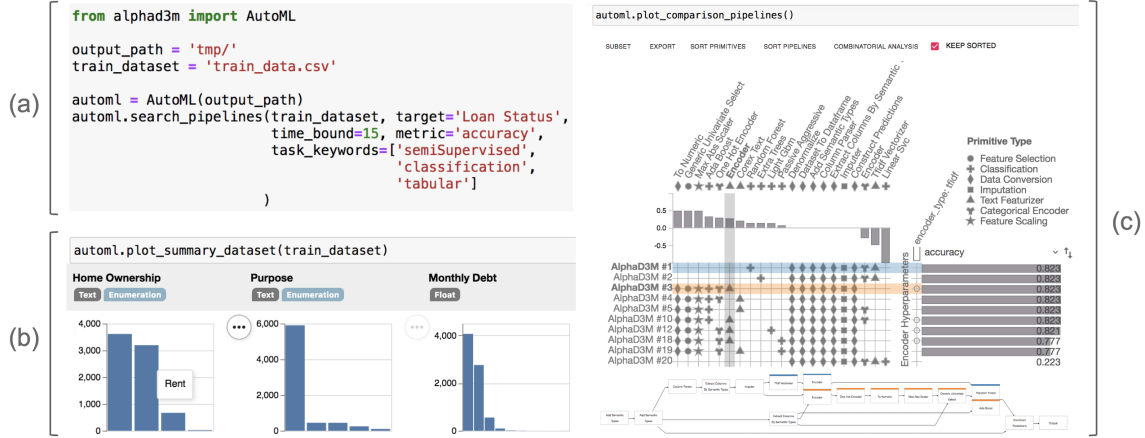


Figure 3: (a) A code snippet to solve a semi-supervised classification task. (b) AlphaD3M allows users to inspect the contents of the input dataset, including column statistics and data types. (c) Analyzing ML pipelines through the integration with PipelineProfiler.

each iteration until a condition (number of iterations) is met. The acquisition function is designed to normalize the performance metric used to synthesize the pipelines between zero and one, as the pipeline execution evaluations increase, the acquisition function gets closer to zero. SMAC requires a set of unique parameters to assign values during its tuning procedure. Since AlphaD3M considers multiple primitives with identical names, it constructs an internal hierarchical nomenclature of parameters and designs their dependencies using ConfigSpace.

C. The API

We have developed a Python-based API that supports the process of building and exploration of ML pipelines within a Jupyter Notebook environment. The API is integrated with the D3M AutoML systems and supports various dataset formats such as raw CSV, D3M, and OpenML. Model synthesis can be done with a few lines of code, as shown in Figure 3(a). The API allows users to (a) define a problem, (b) explore summaries of their input dataset, (c) summarize the produced pipelines and (d) analyze and compare pipelines with respect to their performance scores and prediction outputs. We describe the main components of the API below.

i) Problem Definition To build a predictive model, AlphaD3M needs a problem specification that describes a prediction problem, specifically: (a) the training dataset; (b) a target variable, i.e., what should be predicted by the predictive model; (c) the maximum running time that controls how long the search can take (to control the use of computational resources); (d) the desired performance metric; and (e) a list of task keywords that specify the kind of prediction task and, therefore, the techniques that should be used to solve the prediction problem. Figure 3(a) shows an example of how to define a problem in AlphaD3M.

ii) Data Exploration To build good predictive models, it is important to identify data attributes that lead to accurate predictions. The API provides multiple tools for data exploration. For example, it shows different visualizations (compact, detail, and column views) that summarize the content of tabular datasets (see Figure 3 (b)).

iii) Pipeline Summary After the pipeline search is complete, users can display a leaderboard, train individual pipelines with the complete data, perform predictions and evaluate them against a held-out dataset.

iv) Pipeline Exploration Users can analyze the produced pipelines using the PipelineProfiler [96], which is fully integrated into AlphaD3M as shown in Figure 3(c). PipelineProfiler is a visual analytics tool that enables users to compare and explore the pipelines generated by the AutoML systems.

```

automl = AutoMLClassifier(time_bound=1)

# The feature-engine library is compatible with Sklearn, we just need to import it
from feature_engine.selection import SmartCorrelatedSelection, DropHighPSIFeatures

scs_object = SmartCorrelatedSelection()
dhp_object = DropHighPSIFeatures()

# Adding SmartCorrelatedSelection and DropHighPSIFeatures to AlphaAutoML
automl.add_primitives([scs_object, 'FEATURE_SELECTOR'], (dhp_object, 'FEATURE_SELECTOR'))

```

Figure 4: A code snippet to add primitives to Alpha-AutoML on the fly.

v) Pipeline Refinement and Deployment AlphaD3M allows users to save and load pipelines, enabling users to reload them later and perform analyses without having to re-run the AutoML search. They can load the saved pipelines at any time for training or testing purposes. In addition, users can export pipelines to Python code. This gives them more control and the ability to modify (and customize) the automatically generated pipelines (e.g., change hyperparameters, or replace a classifier primitive). More information about the API can be found on the documentation web page: <https://alphad3m.readthedocs.io/en/latest/api.html>.

3.1.2 Alpha-AutoML

Although AlphaD3M’s approach is primitive-agnostic, it was developed to comply with the D3M architecture and used D3M primitives to build ML pipelines. We have recently developed the next generation of the system, which we call Alpha-AutoML. Alpha-AutoML is an extensible open-source AutoML library that is *pip* installable and it is already available on PyPI⁹.

Alpha-AutoML leverages the deep reinforcement learning component of AlphaD3M, but it relies on standard, open-source infrastructure to specify and run pipelines. It is compatible with state-of-the-art ML techniques: by using the Sklearn pipeline infrastructure, Alpha-AutoML is fully compatible with other standard libraries like XGBoost, Hugging Face, Keras, PyTorch. In addition, primitives can be added on the fly through the standard Sklearn’s fit/predict API, making it possible for Alpha-AutoML to leverage new developments in ML and keep up with the fast pace in the area.

Figure 4 shows how easy it is to add primitives to Alpha-AutoML. To add new primitives, users have to call the method *add_primitives*, sending as parameters the class instances of the primitives and the primitive types. The main goal of this new feature is to allow users to add their custom primitives to the Alpha-AutoML’s search space.

Preliminary experiments suggest that AlphaAutoML is 20x faster than the existing, state-of-the-art D3M systems, producing high-quality models at a fraction of the time. AlphaAutoML has important benefits compared to other AutoML systems: it is extensible making it possible to integrate new developments in machine learning primitives and models; and by relying on widely-adopted open-source de-facto standards, it is easy to maintain.

3.2 Exploring and Interacting with Models

While there is great demand for machine learning (ML) applications, there is a scarcity of data scientists capable of building such models. Automatic machine learning (AutoML) approaches such as Alpha-AutoML and AlphaD3M (Section 3.1) have been proposed to help with this problem by synthesizing end-to-end ML data processing pipelines. However, these follow a best-effort approach and a user in the loop is necessary to curate and refine the derived pipelines. Since domain experts often have little or no expertise in machine learning, easy-to-use interactive interfaces that guide them throughout the model-building process are necessary. In this project, we designed Visus, a system that supports the model-building process and curation of ML data processing pipelines generated by AutoML systems. In Section 3.2.1, we describe the framework used to ground our design choices, a usage scenario enabled by Visus, and discuss the feedback received in user testing sessions with domain experts.

⁹<https://pypi.org/project/alpha-automl>

ML experts also benefit from AutoML systems which can streamline the model construction process. However, these systems operate as an opaque box, the complexity of the underlying algorithms, and the large number of pipelines they derive, they are difficult for developers to debug. In this project, we also implemented PipelineProfiler [96], a visual analytics tool that enables the exploration and comparison of end-to-end ML pipelines produced by AutoML systems.

3.2.1 Visus

Machine Learning (ML) models are increasingly being adopted in many applications. However the supply of data scientists capable of building such models has not kept up with the demand [30]. Recent approaches to automatic machine learning (AutoML) help address this problem by automating not only model fitting, but also the synthesis of end-to-end ML data processing pipelines, from data loading, data cleaning, and feature engineering to model fitting and selection, and hyper-parameter tuning [48, 65, 93]. Commercial solutions such as Google’s Cloud AutoML [79] and Microsoft’s Azure Machine Learning Studio [84] are also available. AutoML systems require as input only the training data and a well-defined problem specification that describes the target variables to be predicted, the data attributes to be used as features, and the performance metric to be optimized. By automating many of the steps of the ML model building process, AutoML systems allow data scientists to solve problems more efficiently.

However, AutoML systems are not sufficient to democratize ML to wider audiences. To empower domain experts, who have little or no expertise in ML, to develop ML pipelines easy-to-use interactive interfaces are needed that guide them throughout the model building process [21, 64]. Additionally, AutoML systems put their best effort to generate pipelines that optimize a given performance metric, which often result in black-box models that cannot easily be understood by humans. This, coupled with the ever-increasing concerns about the societal impact of data-driven decision making [70, 132, 141], underscores the importance of bringing the human in the loop of the model building process. Enabling domain-experts to build ML models not only has the potential to lessen the demand for data scientists, but it also enables experts that possess domain knowledge to assess, validate, and potentially improve model outcomes.

Prior research in the field of visual analytics — which has been regarded as a grand challenge [31] — has proposed multiple frameworks and systems that integrate ML models and interactive visual interfaces to facilitate analytical reasoning [21]. However, these systems typically have to integrate tightly with the model they use to provide insights about the model behavior. With the inclusion of an AutoML system, we cannot assume any prior knowledge about inner-workings of the returned models because of the large variety of components present in the derived ML pipelines.

We propose Visus [110], a system that implements a new framework that supports the model building process and curation of automatically generated predictive data processing pipelines. Similar to prior work [21], our framework is iterative and includes multiple components that support exploratory data analysis (EDA), problem specification, model generation and selection, and confirmatory data analysis (CDA). Additionally, we propose new components that allow users to 1) search the web for new relevant datasets to augment the input data, and 2) summarize, compare, and explain models generated by AutoML systems. The first component enhances the original data by adding new records (through union operations) or new attributes (through joins) that can potentially improve the model performance. The second component allows users to better understand the automatically generated pipelines and to select the one that best fulfills their intent.

A. Proposed Framework

Figure 5 shows the main components of our visual analytics framework. Each rounded box denotes a task and the connecting arrows denote the user workflow. In what follows, we describe each of the components in detail.

Problem Specification The first challenge we address in our framework is the translation of a problem in the user domain (e.g., *What are the important factors that lead to traffic accidents?*, or *What genes or environmental conditions influence a plant’s growth?*) to a machine-readable problem specification that can be fed to an AutoML system. A problem specification consists of a set of well-defined properties that describe the users’ analysis goal: a problem type (e.g., classification, regression, clustering, time series forecasting, etc.), a target attribute to be predicted (e.g., a categorical attribute containing classes, a numerical attribute with values), a performance metric to be optimized (e.g., precision, recall, F_1 , mean squared error, etc.),

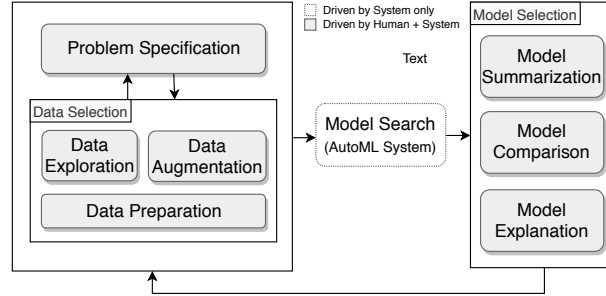


Figure 5: Our proposed framework encompasses problem definition, model generation, and model selection.

and a list of data attributes to be used as features. Additionally, it may contain other parameters such as time constraints (i.e., for how long the pipeline search should run), the maximum number of pipelines to be generated, the performance metrics to be reported and the evaluation method for computing them (e.g., k -fold cross-validation, hold-out test set).

Data Selection In order to create a problem specification, the user first needs to select the data required to solve the problem. There are three data-related components that support the problem specification task: *data exploration*, *data preparation*, and *data augmentation*. We arrange them in a group, namely “Data Selection”, because they are intertwined, in the sense that not only they support the problem specification task, but also are influenced by problem specification. For example, consider the *data exploration and preparation* components. In order to specify a problem, a user might need to first perform an exploratory data analysis to better understand the data. During the course of exploration, the user may realize that an attribute contains outliers which need to be removed, or that an attribute contains missing data that should be filled out with a default value. The user may also decide to update the problem specification to indicate that some attributes should be excluded from the model.

The *data augmentation* component allows the user to search for relevant datasets that could potentially supplement the original data with new records or attributes and, thus, improve the model performance. Describing in details the inner-workings of the dataset search engine we use to support augmentation is out of the scope of this paper. However, it is important to note that it is critical that the search results returned by the dataset search engine should not only be relevant to the domain of interest, but also should be appropriate for integration (via join or union operations) with the original dataset. The main role of the visual interface is then to allow the user to interact with the search results and facilitate dataset relevance prediction (i.e., the search interface should display effective dataset summaries that allow for the user to decide whether the dataset discovered is relevant and should be merged to the original dataset).

Model Search After the data is selected and the problem is specified, the AutoML system can be invoked to carry out the pipeline search phase. Note that this is the only task in our framework which there is no human engagement.

Model Selection AutoML systems aim to generate a model that optimizes a given performance metric in the training dataset. However, the best-scoring model learned using the training set is not always the best model. When the training examples are not diverse and are not good representatives of the real sample distribution, the learned models may not generalize well for samples observed in real-world, and thus the performance might drop in a held-out validation set. Additionally, some evaluation metrics may have trade-offs. For instance, consider the pipelines generated for a binary classification problem. The best pipeline in terms of precision may have a low recall score. Instead of blindly selecting only the best performing pipeline — and given that often a large number of models is evaluated during the optimization process — a better approach for model selection could be to collect a sample of the best-performing pipelines and then present them to a domain-expert to pick the best one. Unlike traditional visual analytics approaches, a system that includes an AutoML component must be able to handle a large number of ML pipelines. Therefore, in order to support model selection, the user interface should provide visualizations to facilitate exploration, comparison, and explanation of multiple models. Our framework includes three components to assist in this task as detailed next.

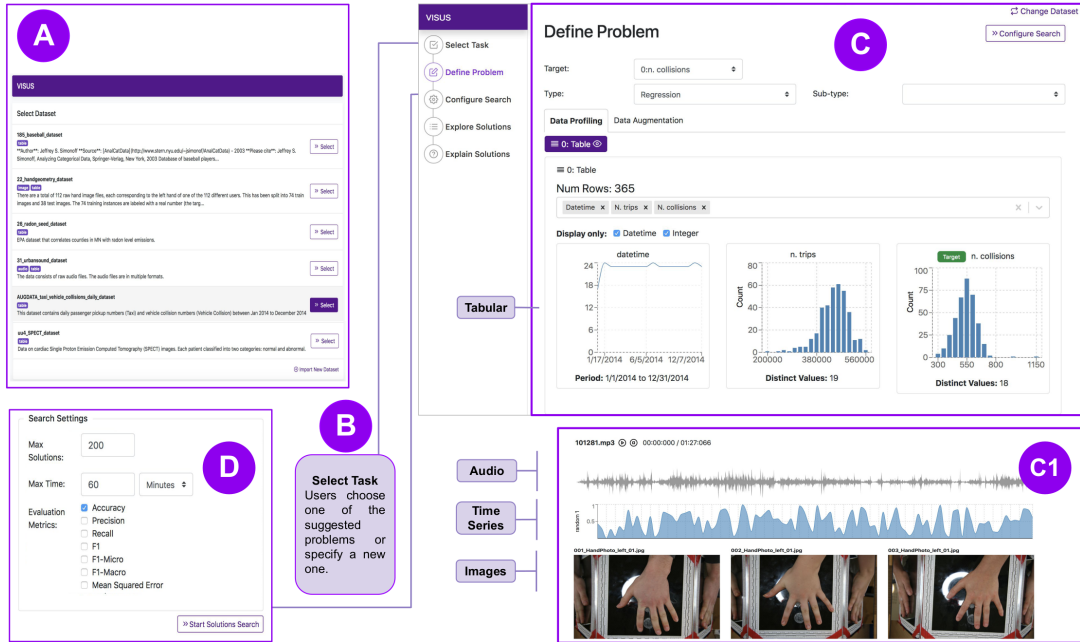


Figure 6: Visus’s data selection and problem definition screens: (A) select or load dataset view, (B) select task view (create a new problem, or load an previously created problem), (C) define problem view (displays data summaries and allows selection of problem parameters), (D) configure search view (allows setup of additional model search parameters).

Model summarization. Since AutoML systems create many pipelines, the end goal of model summarization should be to facilitate the selection of a smaller set of pipelines for further inspection. Visualizations should aim to provide concise representations of a set of pipelines and the ability to easily select the best-performing ones.

Model comparison. Once a sample of the best-performing pipelines is selected, the user is able to compare them in detail. Visualizations should highlight the commonalities and differences between two or more pipelines. More specifically, pipelines may differ not only in performance metric scores but also in their data processing steps, and how they make correct or wrong predictions.

Model explanation. The user might also want to gain a better insight into how models make predictions and how the prediction accuracy is distributed conditioned to specific attributes or the target variable. For instance, one might be interested in understanding what is the impact of certain feature combinations to a prediction outcome. Therefore, explanations might help not only to validate whether the model is making correct predictions for the right reasons but also to learn about data relationships that were not known a priori.

B. Visus User Interface

Visus implements our proposed framework and integrates visual analytics techniques in the model building and curation process. With Visus, we aim to aid domain experts to interactively formulate prediction problems, build models, and select a suitable model among a set of automatically generated models. Visus provides an easy-to-use interface where users can 1) define a problem, 2) explore summaries of their input dataset, 3) perform data augmentation, 4) explore and compare models with respect to their performance scores and prediction outputs.

Define Problem and Explore Dataset As Figure 6 illustrates, the operations for problem specification and data selection are implemented in four screens (views) in Visus: *Select Dataset*, *Select Task*, *Define Problem*, and *Configure Search*.

Select Dataset. Allows the user to select or load a dataset (Figure 6(A)).

Select Task. Users can either create a new problem specification or select one of the existing problems specifications that have been created previously (Figure 6(B)).



Figure 7: Explore models view

Define Problem. To construct good predictive models, it is important to identify data attributes (features) that lead to accurate predictions. For this, Visus profiles the data and displays visualizations that summarize the information, helping users to obtain insights about the features. Visus supports multiple types of datasets, including tabular data, time series, audio, and images (see Figure 6(C1)). For all of them, Visus provides a *select input control* for adding/removing data on demand with auto-complete and multi-select functionalities to help users to easily find data attributes or files.

For tabular data, Visus generates visualizations that adapt to different attribute data types. For instance, for numerical and categorical attributes, Visus generates descriptive statistics and summarizes data values using a histogram. When visualizing time series datasets, Visus provides a stacked time series plots along with an interactive *brushing* tool to help users to explore and visualize different time ranges. For audio, Visus lets users reproduce the audio and display its waveform, and for images, it uses thumbnails to show a sample of the images present in the dataset.

Configure Search. In this screen, users set up additional parameters of the model search such as a maximum number of solutions (pipelines) to be generated by the AutoML system, a maximum wait time to conclude the model search. Furthermore, users can also specify the performance metrics they want to use for evaluating the solutions generated.

Inspect Model Summaries To allow exploration of the pipelines generated by the Model Search, Visus displays them in a solution table as shown in Figure 7(E2). In this table, users can see the different solutions and their associated performance metrics. Users can also sort solutions by metric, allowing them to quickly identify the best solutions according to each metric. Additionally, Visus provides a histogram of the scores associated with each performance metric, which allows the users to visualize the distribution of scores of all generated solutions (see Figure 7(E1)). When the user hovers over a solution in the table, the bin associated with the score of the hovered solution changes its color, allowing quick comparison of its performance against all other solutions. Furthermore, Visus provides a parallel coordinates chart, as shown in Figure 7(E3). Each vertical line corresponds to a performance metric and each horizontal line corresponds to a pipeline. Such a visualization allows for quick identification of metric trade-offs. If the user hovers over a pipeline, he/she can see how its performance measures compare to those of other pipelines. Finally, users can also inspect the actual data processing steps of each pipeline, including their learning algorithms and pre-processing primitives (see Figure 7(E4)).

Examine Model Explanations To better understand how a given pipeline performs, Visus generates more detailed explanations. For classification problems, Visus currently supports two visualizations: a standard confusion matrix and rule matrix [86]. The confusion matrix shows the predicted classes as columns and the true classes as rows as shown in Figure 8(F2). The rule matrix uses a surrogate model to learn classification rules that explain the behavior of the model. It displays rules as rows and features as columns (see Figure 8(F1)). To explain regression problems, Visus supports two visualizations: a partial dependence plot [71]

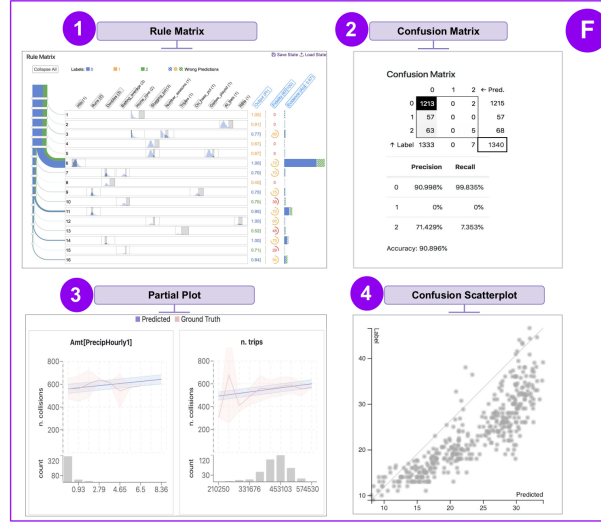


Figure 8: Model explanation support for classification models: (1) rule matrix plot and (2) confusion matrix, and for regression models: (3) partial dependence plot and (4) confusion scatter plot.

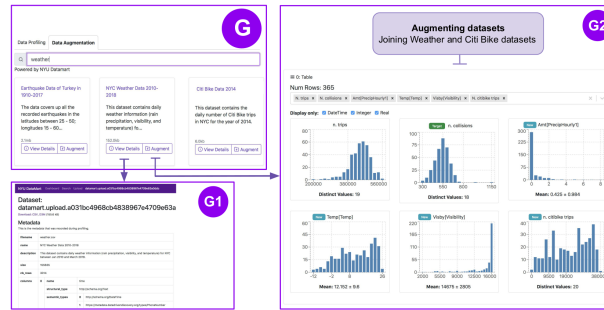


Figure 9: Data augmentation support in Visus. (G) Dataset search results page, (G1) dataset details, and (G2) define problem screen showing data attribute summaries after data augmentation.

and a confusion scatter plot. The partial dependence plots show on the bottom the distribution of samples for each attribute value range, and on the top the predictions and ground truths for these values (see Figure 8(F3)). The confusion scatter plot encodes each data sample as a dot, the predictions in the y -axis, and the value in the x -axis. Figure 8(F4) shows an example.

Data Augmentation In the *Define Problem* view (see Figure 6(C)), the user can navigate to the *data augmentation* tab, where he/she can search for related datasets that could be joined with (join operation) or appended (union operation) to the original data (see Figure 9(G)). Before performing augmentation, users can view details about the suggested datasets by pressing the *View Details* button, as shown in Figure 9(G1).

3.2.2 PipelineProfiler

In addition to Visus, we also developed PipelineProfiler, an interactive visualization tool that allows the exploration and comparison of the solution space of ML pipelines produced by AutoML systems in a Jupyter Notebook environment. This enables complex analyses to be performed over pipeline collections that leverage the rich ecosystem of Python tools for data science.

Figure 10 shows PipelineProfiler being applied to compare pipelines derived by five distinct AutoML systems for a classification problem. 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

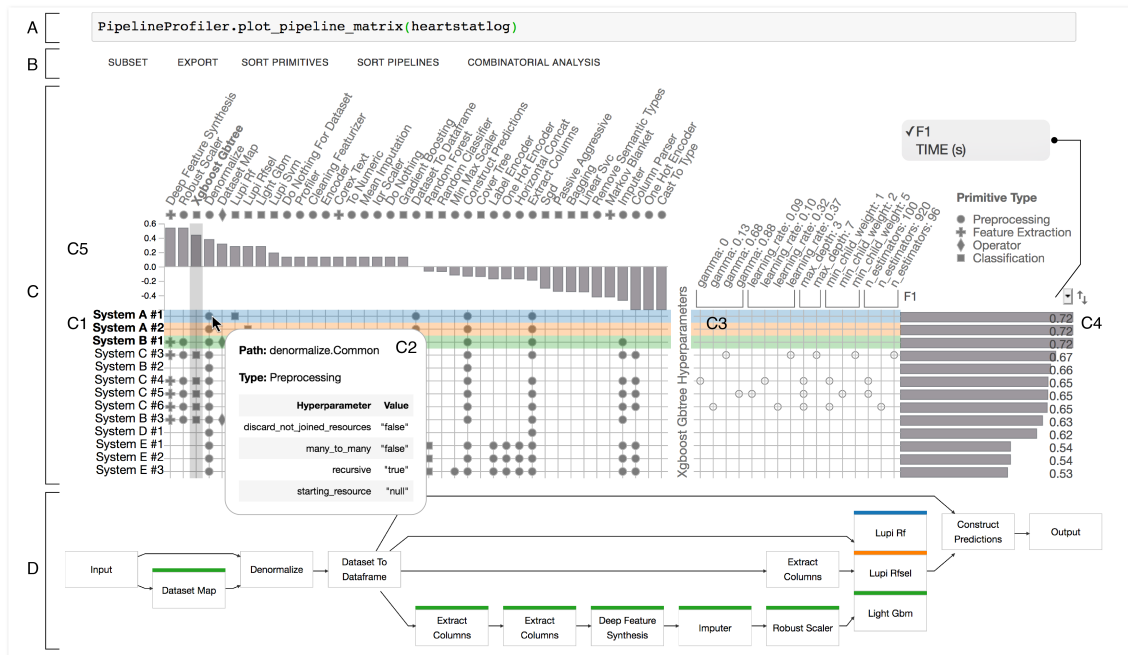


Figure 10: PipelineProfiler applied to the analysis of binary classification pipelines generated by five different AutoML systems.

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).

PipelineProfiler is implemented as a Python library. The front-end is implemented in Javascript with React [54], D3 [17] and Dagre [29]. The back-end, responsible for data management, graph merging and the Jupyter Notebook hooks is implemented in Python with Numpy [123] and NetworkX [68].

The PipelineProfiler library takes as input a Python array of pipelines in the D3M JSON [85, 119] format, and plots the visualization in Jupyter using Widget hooks. The library also enables users to import pipelines from Auto-Sklearn [59]. 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 (Figure 10(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.

3.2.3 Understanding the Requirements for Model Interpretability

To ensure that interpretability research is addressing important needs and providing realistic solutions, it is crucial to understand how the concept of interpretability is being practiced in industry. In order to bridge this gap, we conducted 22 semi-structured interviews with industry practitioners[73]. These interviews aimed to gain insight into how they approach and prioritize interpretability while planning, building, and using their models. The study analyzed the interpretability roles, processes, goals, and strategies within organizations that heavily rely on ML models. The findings indicated that interpretability work frequently involves cooperation and mental model comparison between people in different roles. The aim is to build trust not only between people and models but also between people within the organization.

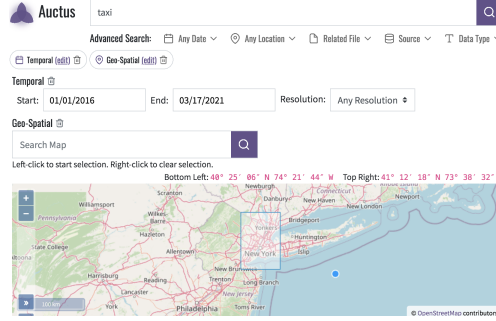


Figure 11: Searching for datasets that mention “*taxi*” and contain records within the NYC area for the 2016-2021 period.

3.3 Data Discovery for Augmentation

While traditionally, machine learning researchers and practitioners have focused on improving model architectures and fine-tuning algorithms, these refined models often fail to produce trustworthy results when deployed and run over new, unseen data. Data-Centric AI [106] aims to address this challenge by focusing on the data used to train the models, since the quality of a machine learning model depends heavily on its training data. In this project, we focused on one aspect of data-centric AI: tabular data augmentation.

Opportunity: Data Abundance. With the push towards transparency and openness, scientists, governments, and companies have been increasingly publishing structured datasets on the Web [1, 36, 38, 52, 87, 92, 125, 128, 133]. As of 2020, Google Dataset Search indexed over 30 million datasets [9]. The availability of these data creates opportunities to answer many important scientific, societal, business questions, as well as to augment data to improve machine learning models.

Challenge: Data Discovery. While data are abundant, finding *relevant data* is difficult. Data are spread over a large number of sites and repositories. Recognizing this challenge, a number of approaches have been proposed to *organize and index data collections* [27], from domain-specific repositories such as NYCOpenData, which collects datasets from the various NYC agencies [92], general data portal infrastructure and data lakes [38, 116], to Google Dataset Search, which indexes a wide range of datasets published on the Web [18]. While these present a significant step towards simplifying data discovery, they have an important limitation: they only support simple, keyword-based search queries over published dataset metadata. This greatly limits a user’s ability to express information needs. In addition, published metadata is often incomplete, and in some cases it can be inconsistent with the actual data. Thus, relying solely on the metadata also limits the discoverability of datasets.

Our Approach. We developed Auctus, an *open-source data set search engine that was designed to support data discovery and augmentation* [4, 22]. Auctus supports a rich set of discovery queries: in addition to keyword-based search, users can specify spatial and temporal queries, data integration queries (i.e., searching for datasets that can be concatenated to or joined with a query dataset), and they can also pose complex queries that combine multiple constraints, as shown in Figure 11.

These queries are enabled in part by a data profiler [37] that we developed to extract useful metadata from the actual datasets. This includes not only summaries (or sketches) of column contents, but also their data types. Users can explore large dataset collections through an easy-to-use interface that guides them in the process of specifying and refining complex queries.

3.3.1 The Auctus Dataset Search Engine

The high-level architecture of Auctus is depicted in Figure 12. In what follows, we describe its key components.

Data Ingestion. Besides allowing users to upload their own data sets, Auctus also makes use of plugins to retrieve data sets from repositories using their APIs. This makes the system extensible and able to ingest data from many different sources. Currently, Auctus supports Socrata [116] (a platform for open government data), NOAA [89] (publishes climate and weather-related data sets), Zenodo [133] (open-access data repository), the World Bank Open Data [128] (datasets on global development), among others.

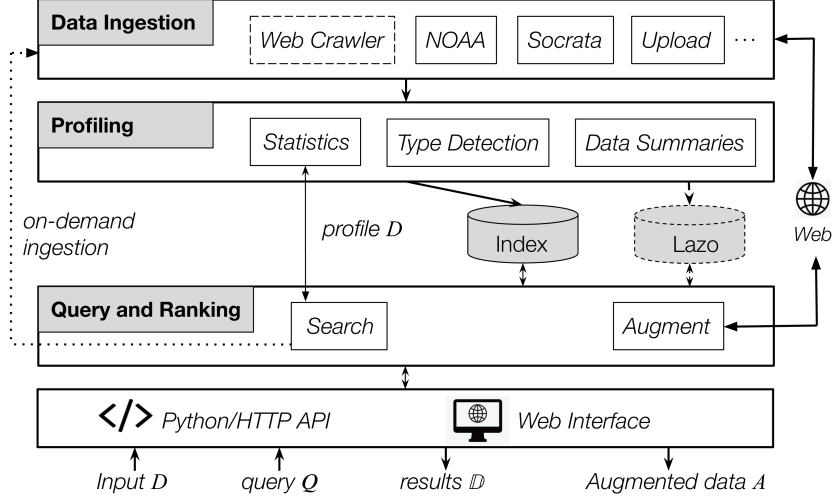


Figure 12: Auctus System Architecture.

Data Profiling. Once data sets are ingested, Auctus profiles them to infer relevant metadata that will be indexed to support discovery queries as well as to construct dataset summaries for result presentation. The profiler performs different tasks, including: *type detection*, i.e., it detects whether columns correspond to categorical, numerical, spatial, or temporal attributes; type-dependent *statistics computation* (e.g., frequency of values, mean, and variance for numerical values), and *data summarization* (see below).

Storing Data and Data Summaries. Auctus stores dataset summaries instead of the full datasets. These summaries are concise and sufficient to create the indices required to answer all queries supported by the system. This choice makes the system scalable and able to handle large, distributed dataset collections.

Currently, Auctus creates summaries for categorical, numerical, spatial, and temporal attributes. The data summaries generated by Auctus are represented by the ranges of their corresponding attributes. During the search phase, Auctus uses these summaries to estimate the size of the intersection between two attributes, concluding whether a join is feasible. To estimate join intersections, Auctus captures fine-grained ranges for data attributes by using the clustering algorithm *k*-means. This strategy produces the desired results while also being efficient. Auctus also stores provenance information to enable the retrieval of the datasets. This allows the system to perform augmentations, and users to download the datasets. Auctus can also cache datasets for efficiency purposes.

Indices. After the metadata are generated, including data summaries, they are indexed in an Elasticsearch [49] server. Numerical and temporal summaries are indexed using range data types, and spatial summaries are indexed using geo-shape data types. To support joinable dataset search, we use Lazo [23], which is a method for set-overlap search based on MinHash sketches and locality-sensitive hashing (LSH), to build an index for categorical attributes.

Querying and Ranking. Auctus supports queries that combine multiple constraints including keywords, temporal, spatial, data type, and data source. The system also supports *data integration queries*: Given an input dataset D_Q , Auctus allows the user to search for datasets that can be concatenated to or joined with D_Q . The data set D_Q must first be uploaded using a provided API or selected from the set of indexed datasets. The system will then generate a dataset profile which is used to probe those indices that support join and union. Finally, the lists of matching datasets from different indices are merged, ranked, and returned as search results.

Join Search. To find other data sets that can be joined with D , Auctus first searches, for each attribute a of D , which other attributes, in the index that corresponds to a 's data type (e.g., temporal attributes searched in the Elasticsearch range index, or categorical attributes probed against the Lazo index), have summaries intersecting the summary of a . Every data set that has at least one intersecting attribute is a potential join result.

Union Search. To find data sets that can be concatenated with D , the indices are searched for any data set that has attributes with the same data types present in D , as well as similar names. Name similarity is

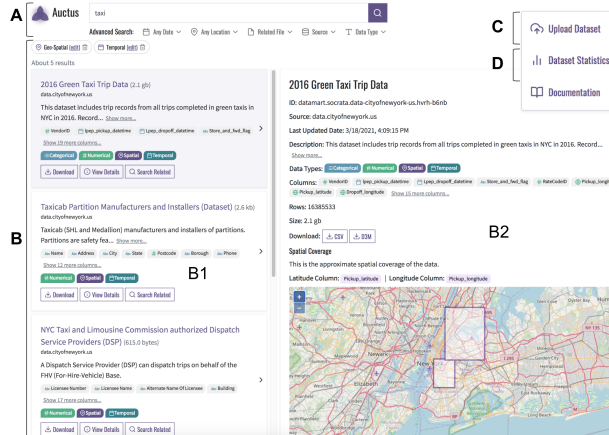


Figure 13: Components of the Auctus user interface: keyword and filter-based search box (A); search results (B); dataset-snippets (B1); dataset summary (B2); dataset upload (C); dataset collection statistics (D).

computed with the *fuzzy query* in Elasticsearch. Results from union searches may match only a subset of D 's attributes.

Ranking. The results of join and union searches, after being filtered based on query Q , are ranked and returned as $\mathbb{D}$. Join results are ranked based on the intersection of the summaries (data sets with higher intersection are ranked higher), and union results are ranked based on the Levenshtein similarity between the names of the matching attributes.

Augmentation. Besides providing search capabilities for data augmentation, Auctus also performs the actual augmentation. Users can choose a data set $R \in \mathbb{D}$, and Auctus will materialize it (using the provenance annotated in the metadata) and perform the join or union operation with D , returning the new, augmented data set A . If multiple attribute pairs match between R and D for a join operation, users can choose which pair(s) they want for the join. For temporal and spatial joins, the attributes are translated into the same resolution before the augmentation.

3.3.2 Auctus User Interface

Auctus provides an easy-to-use interface where users can query for datasets, explore search results including data exploration and augmentation options, explore ingested datasets, and upload new datasets.

Data Discovery Queries. Users can query indices by specifying keywords and constraints using various filters (see Figure 13(A)).

Temporal and Spatial Search. Users search for datasets by specifying a date range (Figure 11) – datasets containing a temporal column that overlaps with that range will be retrieved. They can refine the search by specifying a temporal resolution (e.g., year). To perform a spatial search, users can either draw a bounding box around a geographical area on the map (Figure 11), or specify an administrative area, which Auctus translates into a polygon. Datasets containing spatial attributes that overlap with the search polygon are returned.

Source filter. The source filter allows users to restrict the data sources of interest, and only results from the selected sources are retrieved.

Data Type filter. The data type filter allows users to search for datasets based on the types of their attributes, e.g., categorical, numerical, spatial and temporal, which were inferred by the profiler.

Data Integration Queries. The related file filter allows users to find datasets that can augment a given input dataset (see Figure 14(a)(A)). The user can upload the input dataset or select a dataset from a set of search results. Result snippets for data integration queries include an “augment options” button, that allows users to request and customize the augmentation to be performed by Auctus.

Result Presentation and Exploration. Unlike Web documents which can be summarized using short text snippets, datasets have many different facets that the user must consider to determine their relevance. Thus,

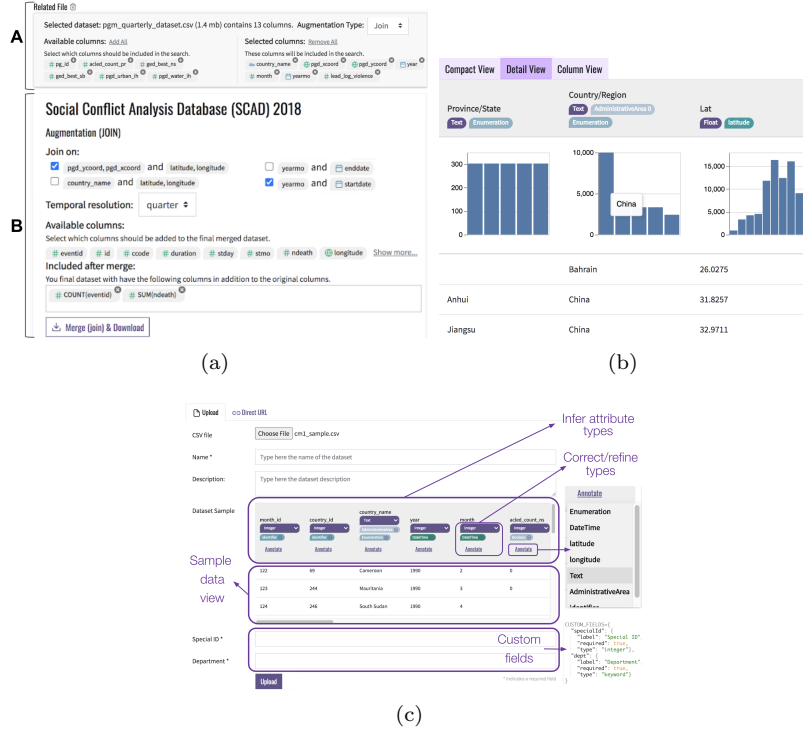


Figure 14: (a) A data integration query in [A], and the augmentation interface in [B]. (b) Data Summary views. (c) Upload page.

an important challenge for Auctus is how to present search results. Figure 13 shows the results for the query specified in Figure 11. On the left, there is a list of search results displayed as snippets (Figure 13(B1)). Users can select a dataset to inspect its details (Figure 13(B2)), which include description, source, attribute names and types, as well as a summary of the dataset’s contents. For spatial datasets, a visualization showing the geographical extent covered by the dataset is displayed. The summary also includes a sample of the dataset records and statistics about its columns (Figure 14(b)). The tabs above the dataset sample allow for the visualization of these statistics in different levels of detail. For example, *Detail View* shows the columns’ value distributions.

Augmentation. If the user provides an input file through the Related File filter, the snippets display a button “Augment Options” under each result on the left. If the user clicks on this button, the augmentation interface will be shown (see Figure 14(a)(B)). If the result is a union, users can select matched pairs of columns, and the columns from the result (right) will be appended to the matching columns of the input data (left). If the result is a join (an example is shown in Figure 14(a)(B)), users can select the columns to be matched at the top of the augmentation screen. Those are the “join keys” that determine which records from the input data should be matched with records from the selected result. Under this area, there is an interface to include columns from the selected result into the augmented table. Users can grab the desired columns from the “available columns” area, drag them to the “included after merge” box, and drop them over the aggregation function they wish to use for that column.

Uploading Data and Curating Metadata. Users can add new datasets to Auctus. After loading a data file, Auctus automatically infers its data types with our datamart-profiler library [37]. As any method for type inference, our profiler is not fool-proof and can derive incorrect results. To address that, Auctus enables users to correct data types manually, and to provide additional annotations for the columns. Auctus also displays a dataset sample so that the user can verify if the detected data types are correct, and check the uploaded data. Additionally, Auctus provides support for custom metadata fields (e.g., data source or grid size). Since these fields can vary widely for different applications, we defined a flexible configuration schema that allows users to customize them for different deployments. The upload page is shown in Figure 14(c).

3.3.3 Scalability and APIs

Auctus has been implemented with scalability in mind: the search engine is entirely containerized using Docker [41]. Each data discovery plugin is an independent container, allowing multiple plugins to be executed in parallel. Auctus can also spin up as many profiling and query containers as required in response to load. Our system can be accessed via a Web UI [4] or programmatically via Python and REST APIs.

3.3.4 Discovering Datasets on the Web

Auctus indexes datasets in known repositories and its index is necessarily incomplete. When a user is trying to study a particular domain or question, they must search more broadly for relevant data that can then be indexed by Auctus. Consider the following example.

Explaining Food Insecurity in Ethiopia. Despite the strong economic gains over the past two decades, food insecurity persists in Ethiopia. According to the information from the United Nations World Food Programme, an estimated 8 million people require food assistance and about 3.9 million women and children are nutritionally vulnerable in Ethiopia [102]. A group of data scientists aims to construct predictive models for analyzing the reasons behind the food insecurity problem in Ethiopia. They started with population, health, and income datasets provided by the Ethiopian government. On the Web, there are many additional and potentially relevant datasets, including, for example, indicator data provided by the World Bank and the Food and Agriculture Organization, which cover different aspects of life in Ethiopia – from weather and droughts to conflicts. By collecting and analyzing relevant datasets, domain experts can not only better understand the reasons behind the food insecurity, but also discover new features to make predictive models more robust and accurate. These in turn, can help inform interventions to avoid food insecurity and mitigate its effects.

However, finding relevant datasets on the Web is challenging. There is a large number of datasets spread across many different data repositories and Web sites. As a point of reference, if we consider urban datasets, there are over 200 different repositories, containing many thousands of datasets, that use the Socrata API¹⁰ (e.g., New York City Open Data¹¹, Chicago Data Portal¹²).

Recognizing this challenge, Google took a first step towards improving findability with Google Dataset Search (GDS), a search engine for datasets [18]. However, to have datasets indexed by GDS, data publishers must include metadata using the standard formats such as `schema.org` or W3C’s DCAT [18]. Unfortunately, relying on metadata markup leads not only to a limited coverage but also to a veracity problem: up to 61% of web sites that provide `schema.org/Dataset` markup on their web pages do not actually describe datasets [2]. Therefore, an open question remains for domain experts: how to efficiently discover *all* data relevant to a given domain of study or research question, including datasets not indexed by Google Dataset Search.

One possible approach is to use a focused crawler [7, 24, 26, 39, 40, 83, 113, 122]. Instead of attempting to cover all Web pages, a focused crawler tunes its search strategy to search for a target concept (or topic), while maximizing the number of on-topic pages it retrieves and minimizes the number of irrelevant pages visited. Focused crawlers thus bring many benefits: they lead to substantial savings in hardware and network resources compared to searching the whole Web; they make it cheaper to maintain the crawl up-to-date; crawls can go deeper and obtain a better coverage for a given topic; and the derived index, being focused, is more likely to return a higher fraction of actually relevant pages, reducing the information overload. Focused crawlers have been proven effective to acquire topic-specific Web pages as well as to discover specific objects inside Web pages, such as events [53, 75] and product information [43, 103]. They leverage the page contents to determine relevance to a topic and to harvest links that are used to expand the search. But crawling for datasets on the Web creates new challenges. Datasets are often published in repositories that do not link to other datasets or repositories. Therefore, we need new strategies to find regions of the Web that contain domain-specific datasets. Another challenge comes from the complexity involved in configuring focused crawlers. To specify a topic, an expert must provide both a set of seed URLs that serve as the starting points for the crawl and a *classifier* to automatically recognize relevant content. While finding seeds can be time consuming, the task of creating an effective classifier is difficult and often out of reach for experts without training in computing.

DSDD: A Usable Dataset Discovery Framework. To address these challenges, we proposed DSDD, an

¹⁰<https://dev.socrata.com/>

¹¹<https://data.cityofnewyork.us>

¹²<https://data.cityofchicago.org>

end-to-end system for domain-specific dataset discovery on the Web [135]. Because datasets are not explicitly linked and are distributed over disconnected regions of the Web that cannot be reached by crawling alone, we need mechanisms to *jump* to pages that serve as entry points to dataset repositories. To address this challenge, our system tries to mimic the manual and iterative process users follow to discover relevant datasets: issue queries (e.g., "*Ethiopia dataset*") to a search engine, visit the returned URLs, determine their relevance, and refine the search (e.g., search for links *related* to one of the relevant results). Our system automates this process and utilizes a rich set of discovery actions to *jump-crawl* the Web, including keyword and related search supported by search engines, and forward and backward crawling [25]. However, selecting the most appropriate discovery actions at each iteration is difficult because the effectiveness of an action depends on multiple factors, including the domain of interest and candidate pages. Random selection can be inefficient, because due to the randomness, it is possible to always select an action that has already had bad performance in the past. We propose an algorithm based on multi-armed bandits to dynamically schedule the actions with the goal of attaining a balance between the exploitation of productive regions and the exploration of new regions that can increase the diversity of the discovered datasets.

DSDD attains *usability* by 1) requiring users to provide just a small set of descriptive keywords for the domain of interest; and 2) applying online learning to incrementally and automatically improve a classifier that determines which pages are potential dataset repositories. The classifier is initialized with manually collected samples of general open data portals, to have a sense of what features of a web page document represent a dataset repository. At each iteration, the classifier decides potential dataset repositories, then DSDD verifies if they actually lead to dataset pages and uses the verified results to automatically update and improve the classifier.

Unlike Web pages and objects inlined in these pages, datasets can be large and it may not be feasible to download and parse their contents at crawling time to determine relevance. Even if the data are downloaded, determining their relevance and quality is challenging [98, 99, 124, 131]. Therefore, our goal is to maximize the number of dataset pages (and repositories) we discover for a given domain, so that a rich data collection can be assembled. After the datasets are collected, profiled and indexed by a dataset search engine such as Auctus [22] and Find Open Data [137], users will be able to pose a rich set of discovery queries [136] to identify which datasets are relevant for different tasks (e.g., improve an ML model, augment an existing dataset). Information about datasets found to be relevant can then be used to refine the discovery process of DSDD.

We performed an extensive experimental evaluation in multiple domains, and presented results that show that not only does our approach lead to harvest rates that are nearly 100% or higher compared to state-of-the-art domain-specific content discovery systems, but it also finds dataset pages that are not indexed by Google Dataset Search [135].

3.3.5 Join-Correlation Discovery Queries

The increasing availability of structured data sets, from Web tables [20, 78] and open-data portals (see e.g., [128, 129]) to enterprise data, opens up opportunities to enrich analytics and improve machine learning models through *relational data augmentation*. The challenge lies in finding relevant data sets for a given task. We proposed a new class of data search queries that retrieve related data sets by uncovering correlations between numerical columns in unjoined data sets.

Consider the following examples.

Example 1: Explaining Traffic Fatalities.. Based on the observation that there is a relationship between high traffic speed and the number of fatalities, the City of New York, as part of the Vision Zero initiative [88], reduced the speed limit on its streets from 30 to 25 miles per hour. While the lower speed limit was partially effective, fatalities were still high and there was a need to understand why. Analysts hypothesized that an increased number of bikes and the weather can also lead to more traffic fatalities. By examining data sets published in NYC Open Data [92], they validated their hypotheses: from CitiBike data, they found that when there are more active bikes, there are more accidents; and they also observed that high precipitation is correlated with a larger number of accidents.

Example 2: Improving Taxi Demand Models.. A data scientist at the NYC Taxi & Limousine Commission created a model to predict taxi demand using a data set that contains historical data about taxi trips and their

associated pick-up times and location zip codes. To improve the model, she had to find additional features that can influence taxi demand. Domain experts suggested a set of indicators for her to examine, including weather, major events, and holidays. She obtained the relevant data sets and observed a substantial reduction in the root mean squared error after augmenting the training data.

In both examples, the analysts knew which data sets they needed and where to find them. Often, this is not the case: finding relevant data is a difficult and time-consuming task [55]. On the Web, data are distributed over many sites and repositories, and in enterprises they are stored in a plethora of systems and databases. As a point of reference, internal research at Lyft has found that their data scientists spend 25% of their time on data discovery – they only spend more time (40%) on model development and deployment [67].

Discovering Correlated Data sets Augmentation queries over large data set collections can enable analysts to uncover previously unknown relationships that lead to new hypotheses. In Example 1, to explore the question “What causes traffic fatalities?”, the analyst could search the NYC Open Data [92] repository for data *related* to the traffic fatalities data set, which contains the total number of daily fatalities for each zip code; and in Example 2, she could search for data related to the taxi demand data set, which consists of the total number of hourly pickups per zip code.

For both examples, methods devised to find joinable tables [23, 138, 140] would find relevant results among over 2,000+ data sets in NYC Open Data, but they would find *too many*. Specifically, these methods find columns that have a large overlap with the input data set, so they would return all data sets that contain a column with zip codes in NYC – most of which are not helpful to the users’ information needs. For the examples above, further filtering is needed. To express their information need more precisely, analysts would benefit from asking *join-correlation queries*.

Definition 1 (Join-Correlation Query) *Given a column Q and a join column K_Q from a query table T_Q , a join-correlation query finds tables T_X in a dataset collection such that T_X is joinable with T_Q on K_Q and there is a column $C \in T_X$ such that Q is correlated with C .*

In our first example, the analysts could issue a query to retrieve data sets that join with the traffic fatalities data set and that also contain a column that correlates with the actual number of fatalities; in the second, the data scientist could search for data sets that join with the taxi demand data set and that contain a column that correlates with the actual taxi demand.

The key challenge in evaluating *join-correlation queries* is how to do so efficiently. One approach is to first find joinable tables, and then to explicitly compute correlations between Q and all columns of the discovered tables using, for instance, the Pearson’s correlation coefficient for linear relationships, or the Spearman’s rank correlation coefficient to capture non-linear relationships [104]. However, this approach requires the joinable data sets returned to be downloaded and joined with the query table. When these tables are large, they may not fit in memory and the cost of executing join operations can be prohibitive. Furthermore, some correlation measures are expensive to compute, e.g., to compute Spearman’s correlation the data must first be sorted. This problem is compounded for queries that return a large number of data sets and require many joins and correlation computations to be performed. As a point of comparison, joining a data set on taxi pickups (approximately 1GB) with a data set on precipitation (approximately 3MB) took about 29 seconds and computing the Spearman’s coefficient between the numbers of pickups and precipitation levels in the resulting joined data set took about 5 seconds (on a Intel Core i5 2.4GHz CPU).

Our Approach: Ranking Data sets via Correlation Estimates To reduce the evaluation cost of join-correlation queries, as an alternative to using the entire data, we investigated the use of data synopses to estimate the results. Data-intensive algorithms can often be optimized by reducing the size of the input data with sampling techniques [32], at the cost of obtaining approximate results. In our setting, however, naïvely sampling data to estimate correlations does not work: it is not possible to sample columns to estimate the correlation without first executing the join. To support the *efficient discovery of correlated columns from distinct data sets at scale*, we propose a sketching method for *estimating correlations between columns from unjoined data sets* based on column synopses, namely *Correlation Sketches* [108]. These synopses are constructed using only data from individual columns, and thus they can be *pre-computed and indexed* to support discovery of joinable data sets and *fast correlation estimation* at query time.

Our method constructs a synopsis $L_{\langle K_X, X \rangle}$ for any given pair of columns $\langle K_X, X \rangle$ that belongs to a table T_X , where K_X is a categorical column and X is a numerical column. A pair of synopses $L_{\langle K_X, X \rangle}$ and $L_{\langle K_Y, Y \rangle}$

(for tables $\mathcal{T}_X$ and $\mathcal{T}_Y$ respectively) can be used to estimate the correlation between the numerical columns $X_{X \bowtie Y}$ and $Y_{X \bowtie Y}$, generated *after* joining tables $\mathcal{T}_X$ and $\mathcal{T}_Y$ on columns K_X and K_Y . Note that $\mathcal{T}_X$ and $\mathcal{T}_Y$ are heterogeneous and need not have the same join keys or the same number of rows. As such, our sketching method enables the construction of an index for a large number of tables that can be used to support *both* joinability queries and to estimate the correlation between a query column and indexed columns.

Our sketching method builds upon and extends state-of-the-art hashing techniques [6, 13, 14, 74, 130]. In particular, we prove that our sketches can reconstruct a uniform random sample of the paired columns $X_{X \bowtie Y}$ and $Y_{X \bowtie Y}$. This leads to an important property: besides correlations, our sketching approach can handle *any* statistic that can be estimated from random samples (e.g., entropy and mutual information). While other statistics are out of the scope of this paper, we demonstrate the flexibility of our method by using it to estimate a variety of different correlation coefficients.

We show both theoretically and experimentally that our approach is effective and provides accurate estimates for correlation. Moreover, our analysis provides mathematical tools for dealing with approximation errors typical of sketching algorithms. For join-correlation queries, these errors may lead to false positives: columns are returned which seem more correlated, based on the sketch, than they actually are. This problem is an issue for queries over large data set collections, as there can be many false positives, simply by chance. To address this issue, we derive sub-sample confidence interval bounds to estimate approximation errors. Our bounds are based on simple columns statistics like sample size and data range and, in contrast to prior work [12, 15, 16], do not rely on distributional assumptions. We use these bounds to design a set of scoring functions that rank data sets based on both their estimated correlation with a query data set, and on our confidence in that estimate.

We evaluated our method experimentally using both synthetic and real-world data sets. A comparison of the estimates produced by *Correlation Sketches* with the actual correlation values shows that it derives accurate estimates. In addition, we assessed the effectiveness of different ranking functions that leverage *Correlation Sketches*, and showed that they improve ranking performance up to 193% in terms of mean average precision when compared to a scoring scheme based on overlap size, commonly used for joinability queries [23, 138, 140].

3.4 Results and Discussion

3.4.1 Auctus

A. User Study

We conducted two user studies to evaluate the search engine result page (SERP) of Auctus [22], our dataset search engine designed to support data discovery.¹³ Auctus’ goal is to provide a user-friendly interface that allows users to pose not only simple textual queries that match metadata but also a wide range of structured data queries [56, 108, 109, 139] that match data in the tables (e.g., finding datasets that join with a given input query dataset). For this study, however, we restrict the study scope and focus on evaluating the effectiveness of the visual components from Auctus’ SERP.

We conducted two-round interviews for this user study. The first round of interviews were conducted to validate the Auctus design decisions, and the second round was run to evaluate Auctus usability and design improvements.

A.1. Expert Interviews

To validate our design decisions, we conducted a first round of interviews with seven people (U1-U7). Each interview took 40 minutes and proceeded as follows. We first conducted a tutorial session to present Auctus to the participants and clarified any questions they had (12 min). This tutorial consists of three parts: (1) a video demonstrating our system, (2) a presentation emphasizing Auctus’ functionalities that will be mostly used during the user study, and (3) an interactive exploration with an example query to familiarize the participant with Auctus. Then, we let them answer several dataset search questions using Auctus (23 min). Finally, we asked the participant to fill out a system usability questionnaire and left any comments on the system (5 min). The participants were free to explore any features in Auctus that may help them answer these dataset search questions. They were asked to think out loud while using Auctus. When the participants performed the task, we took notes related to the actions they performed.

¹³Auctus is available at <https://auctus.vida-nyu.org/>.

Dataset Search Questions. To assess and validate our design decisions, we evaluate the effectiveness of the different features of Auctus using a variety of dataset search questions:

- *Effectiveness of Dataset Summary.* Useful information is highlighted in the Auctus' interface to summarize the contents of each dataset: column names and data types displayed in the snippets; additional information like number of rows and dataset size in the dataset summary, which is located on the right side of the interface. We formulate two questions to evaluate the effectiveness of these features: The first asked users to identify datasets that contains certain data types, and the second asked users to ascertain the dataset relevancy in terms of size (e.g., according to their number of features).
- *Effectiveness of Spatial Coverage.* One prominent feature of Auctus is the spatial coverage information, which can be shown through a *Spatial* data type tag or a *Spatial Coverage* map. This can help users quickly find datasets that contain geographical information. We designed a question that asks the participant to explore the search results and check if any of the datasets contains spatial information.
- *Informative Column Statistics.* The data summary views (Figure 14) include statistics about each column and visualizations in different levels of detail. To examine this informative feature, users were asked to answer a statistical question about a specific column in a dataset.
- *Effectiveness of the Entire System.* Finally, we designed a question that is not focused on a specific feature of Auctus. Instead, the participant can use any features of Auctus to answer the question and we can understand which are the *most-frequently used* features. These kinds of questions are formulated as follows: *Which datasets are more relevant to study topic T?*

User Insights and User Feedback. We received positive feedback from the participants. They expressed interests in using Auctus and suggested new features to improve the system. Followings are some of their feedback:

- U1 mentioned that it is useful, for comparison purposes, to have the dataset snippets (left side of the interface) and the dataset summary (right side of the interface) side by side, and also to have repeated information on both sides (e.g. *data types*). He says that he can have the dataset of interest in the right side and then scroll down the left side to check the snippet information of other datasets and make comparisons.
- Most of the participants (U1-U7) appreciate the *Dataset Summary* component (see Figure 13(B2)). They used this side of the interface to actually evaluate whether the datasets was useful for their task or not. However, some of them (U1, U3) suggested to better organize the information presented in the top section of this component: grouping related information may help users to identify relevant data more quickly.
- U1-U5 expressed interests in the *Data Summary Views* (Figure 14(b)), which summarize data samples in three different ways. Most of them found the *Detail View* the most useful tab because it shows summaries and sample data in the same view. Based on this feedback, we rearranged the tabs order to make the *Detail View* the default tab.
- U2 expressed interests in using Auctus for her study on evaluating different machine learning algorithms (e.g. find datasets of different size). For suggestions, she mentioned that, besides number of rows, Auctus should also display the number of columns. This could help her to see how much data contains a dataset in terms of columns. For the histogram plot in *Detail View*, it would be great to sort the bars in a consistent way (e.g. either by frequency or by alphabetical order).
- U5 suggested that, similar to the spatial coverage map, it would be great if Auctus also had a temporal coverage plot to help users better identify the datasets that are in the temporal range they desire.

A.2. Usability Evaluation

After the first round of interviews, we implemented improvements in Auctus based on users' insights and feedback. To evaluate Auctus usability and design improvements, we conducted a second round of interviews with ten people. Two types of usability evaluation techniques were employed: The performance-based evaluation and the questionnaire-based evaluation.

The goals for this second user study are:

1. Explore and analyze the datasets made available for the NTCIR Dataset Search 2 competition.
2. Assess Auctus' ability to help users to complete search tasks. To carry out this performance-based usability assessment, we measured and analyzed observed question success rates, question completion times, features usage ratings, and users' feedback.
3. Evaluate the perceived usability by performing a standardized questionnaire-based evaluation (SUS). This evaluation was performed in both rounds of interviews, and therefore, it allows us to compare Auctus UI's improvements implemented for this second round.
4. Identify system weaknesses by according to the results obtained in the performance-based evaluation. This helps us to recognize which features are hard to understand or not intuitive.

Each interview took 50 minutes and proceeded as follows. We first conducted a tutorial session, as we did for the first round (12 min). Then, we let participants perform the actual user study using Auctus (30 min). The study consists of 3 search queries (each of them containing 5 questions). The queries cover three topics: 1) crime rate, 2) hospital, and 3) causes of death. Each query comes with a background description and a direct link to the searched results in Auctus. Note that we recorded response time for each question separately for later analysis. Finally, we asked the participant to fill out a system usability questionnaire (SUS) and left any comments on the system (5 min). The participants were free to explore any features in Auctus that may help them answer these dataset search questions.

Performance-based Evaluation. Performance evaluation was done based on certain metrics which are measured while performing the questions. These metrics are described below:

- *Question Completion Time.* We designed five questions for each query. Each question asks same type of question across different queries. For example, question 1 in all 3 queries ask about data types of relevant datasets. As a result, for the same type of question, we expect that users will spend less time to find the answer as they become more familiar with both question type and the Auctus UI. For example, less time should be spent on question 1 in query 3 than that in query 1. We record the user response time of each question.
- *Question success rates.* We analyze the accuracy of each question to understand if Auctus can help users find the correct answer to dataset query questions. Accuracy of questions to all three queries are shown in Table 2.
- *Feature Usage.* For each question, we also conduct a feature analysis question to help us better understand which features are most helpful to answer each question.

Based on the analysis of *question completion times*, *question success rates*, *post-question Auctus' features usage ratings for each question*, we obtained the following interesting observations:

- *Completion Time.* For straightforward questions that are easy to solve, a decreasing trend in time is observed Figure 15(a). For questions that are difficult to solve or required a deeper analysis, no obvious decreasing trend in time is observed Figure 15(b).
- *Success Rates and Feature Usage.* Table 2 demonstrates that Auctus can help users easily find correct answers to most questions for all queries. However, some questions were found to be consistently hard across all queries. For example, question 5 has accuracy of 0.8, 0.8 and 0.6 on three different queries. Even though users spent significant amount time on this question (see Figure 15(b)), some users were not able to find the correct answer. Moreover, question 5 in query 3 has the worst accuracy because more efforts are needed to carefully investigate dataset views (i.e. dataset samples and statistic summaries) compared to that in query 1 and query 2, as shown in Figure 16.

Table 2: Question success rates (accuracy).

	Query 1	Query 2	Query 3
Question 1	1.0	1.0	1.0
Question 2	1.0	1.0	1.0
Question 3	1.0	0.9	0.8
Question 4	1.0	1.0	1.0
Question 5	0.8	0.8	0.6

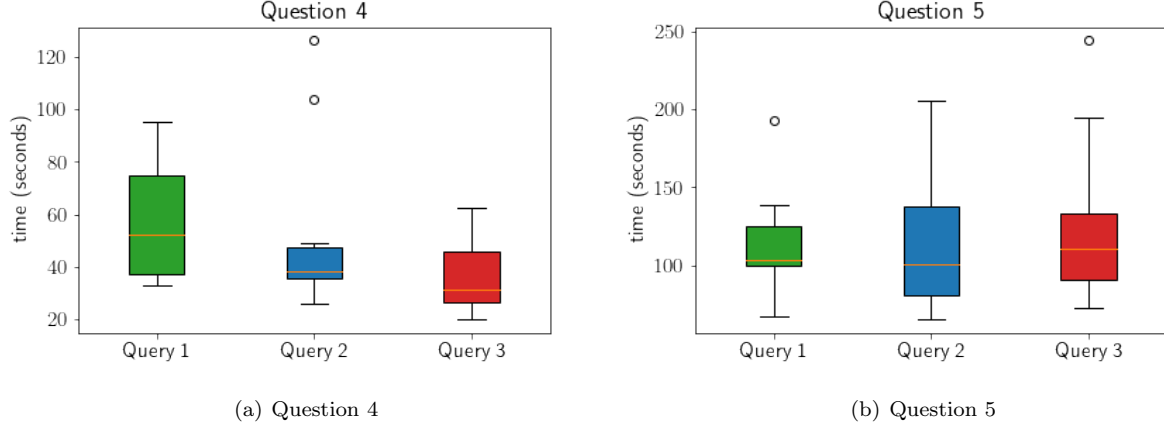


Figure 15: Time spent by users to complete Question 4 and Question 5 for different queries.

- *Feature Usage.* For questions that require to compare multiple datasets, users prefer to use Dataset Snippets. Users mentioned that having summarized information about the dataset in the snippets let them quickly identify datasets with certain characteristics, like data types, as shown in Figure 17. In cases where the users need to inspect column-specific information, they seem to prefer Auctus features that belong to the *Dataset Details* view. For questions that involve analysis of geographical information, most of the users (96.6%) make use of the spatial coverage map.

Questionnaire-based Evaluation. We also evaluated the usability of Auctus using the well-known System Usability Score (SUS) [19]. We conducted a survey at the end of each interview for both user studies: 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 to assess the quality of system interfaces [5]. In the first round of interviews, Auctus obtained an average score of 88.33 ± 15.51 . According to Bangor et al. [5], a mean score above 80 is in the fourth quartile and is acceptable. The average score achieved in the second round of interviews (after improvements) was 88.80 ± 15.45 . Even though this is not a significant increase, the score from the second user study suggests that the changes implemented based on user feedback improved the system usability.

B. Case Studies

In this section we describe workflows that demonstrate how Auctus may be used.

B.1 Case Study: Conflict Forecasting

While analyzing conflict forecasting problems, researchers often use predictive modeling to guide policy-making decisions, and to assess and compare theories of conflict [45]. In this context, it is crucial to identify new data sources, merge those data, and evaluate the contribution of different features. Furthermore, most conflicts materialize as events, and if they occur in heterogenous spatio-temporal levels and need to be analyzed in tandem, data integration can be challenging. Consider that a researcher is studying conflict events in Africa. She uses the grid [101] dataset for Africa with conflict events aggregated into quarterly counts to predict state-based conflicts. Since protest events data can be very useful, and since her grid dataset does not have

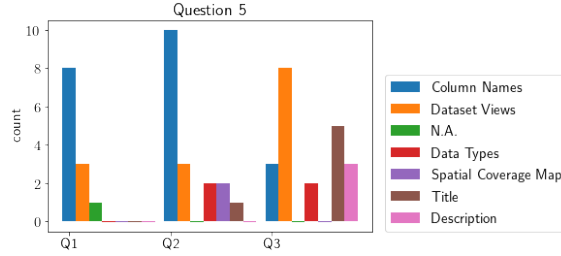


Figure 16: Distribution of different features used on Question 5.

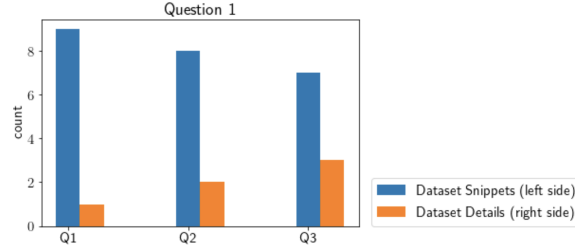


Figure 17: Distribution of different features used on Question 1.

any measure of protests, she decides to use Auctus to discover and join this type of dataset to her input data. She uses "protest" as a keyword, and also uploads the grid dataset as an input query in Auctus. After running the query, the highest ranked result is the Social Conflict Analysis Database (SCAD). To verify its suitability, she explores the dataset through the "View Detail" tab. After inspecting the Summary Data view, she realizes that SCAD contains information not only on protests, but also on other destabilization events. She can also quickly see that it captures events in Africa because of the map visualizations, and also finds spatio-temporal matches. She then checks the possible augmentation options moving in to the augmentation interface. As shown in in Figure 14(a)(B), Auctus automatically detects joinable columns. She then selects the temporal and spatial levels of her interest, and uses aggregation functions to count the number of destabilizing events and to sum the number of fatalities. Next, she presses the augmentation button and all the events in SCAD become aggregated into the grid dataset. The augmented dataset can be used to explore new research questions, such as whether the number of destabilizing events at time t is a predictor for state-based violence at time $t+1$.

B.2. Case Study: Bicycle Usage Prediction.

Predicting the number of daily bicycle trips is an important step towards implementing better policies for this means of transportation in NYC. Consider that an expert from the NYC department of transportation decides to build a model using the East River Bicycle Trips data. This department installed automated counters in all East River bridges, which provide the number of bikes crossing them on a daily basis. This number can be used as a proxy for the overall bicycle usage in NYC. Besides daily bicycle counts, the data also contains maximum daily temperatures for NYC. Unfortunately, her initial dataset only covers bicycle usage for April 2020, which is not very informative. To take into account a larger period of time, she uses Auctus to find compatible data for more months of 2020, which can then be concatenated to her original input data within our system. When using a random forest regressor to predict the number of bicycle trips using daily temperatures as a model feature, she obtains an R^2 score close to 0.25. She is relying on a single feature to make her predictions, so her model is probably underfitting. To improve it, she uses Auctus again to find and augment her current input data with weather information, including daily rainfall levels. The R^2 score then increases to approximately 0.56, which represents a substantial improvement. A video demonstrating this use case is available at <http://bit.ly/auctus-demo>.

B.3. Integration with External Tools: Modeling Political Violence

Besides the web interface (see Figure 13), Auctus can be accessed through a REST API [110] which makes it very easy to integrate with other tools. Next, we demonstrate how Auctus can work alongside external tools such as TwoRavens, [46] a framework for subject-matter experts to extract useful insights via *Human Guided machine learning* (HGML). In doing so, we show how it facilitates improvements in modeling, measurement, and information exploration.

Improved Modeling. Conflict researchers use predictive modeling for a number of reasons, including to guide policy-making decisions, assess and compare theories of conflict, and to evaluate the contribution of novel research methods [45]. For each of these purposes, it is crucial to identify new data sources, merge those data, and evaluate the contribution of different features. This is an iterative process loop where data is evaluated, merged, and assessed. This workflow is facilitated through the integration of TwoRavens and Auctus.

In practice, researchers who study conflict forecasting have to handle the fact that conflict events might occur in a variety of spatially and temporally disaggregate levels, and if they appear in different datasets that need to be analyzed in tandem, joining the datasets may be challenging. For example, the aforementioned ViEWS project forecasts conflicts using PRIO-GRID, which is a 0.5×0.5 -degree spatial grid [120]. Similarly, conflict event data projects such as the Social Conflict Analysis Database (SCAD) regularly identify the events' latitudinal and longitudinal coordinates [107]. If researchers were to join these datasets, they would have to specify functions to identify whether an event falls within a particular grid, if it falls within the grid in the specified time interval, and then use an appropriate function to aggregate cases when more than one event occurs. Auctus has features to accomplish this join of grid-level data with arbitrary event data, also allowing for different levels of temporal specification and different aggregation functions.

In this particular case study, we use grid data for Africa with conflict events aggregated into quarterly counts. Thus, each grid contains four observations for each year. The target variable is again state-based conflict [118]. Our hypothesis is that social unrest and destabilizing events contribute to the use of force domestically [28]. The required steps are described as follows:

- Transition to Auctus from TwoRavens (see Figure 18)
- Search keyword *protest* and merge SCAD into the grid data
 - Specify spatial and temporal levels
 - Select aggregation function to *count* the number of destabilizing events and to *sum* the fatalities
- Transition back to TwoRavens from Auctus
- Forecast state-based conflict

After exploring the results generated by running the AutoML system, we find that the number of destabilizing events at time t is a valuable predictor of state-based violence at time $t + 1$. Joining these data also enables the researcher to use other automated features of the TwoRavens system, including the *discovered problems* component that automatically formulates models to predict features in the data, potentially leading to new research questions. As an example, are there existing variables associated with the merged SCAD data features that are related in an interesting way? Upon merging another event dataset, this system feature would apply to combinations of the original data, the SCAD data, and the newly joined data. Thus, this integration between TwoRavens and Auctus provides compound benefits to the researcher.

3.4.2 Visus

In this section, we describe a usage scenario that illustrates the use of Visus to support data analysis and decision-making processes and discuss the feedback received in user testing sessions.

A. Use Case

Following, we describe a data analysis scenario that Visus enables. Consider the following example. As part of the New York City Vision Zero Initiative [91], an expert from the Department of Transportation is trying to devise policies to reduce the number of traffic fatalities and improve safety in NYC streets.

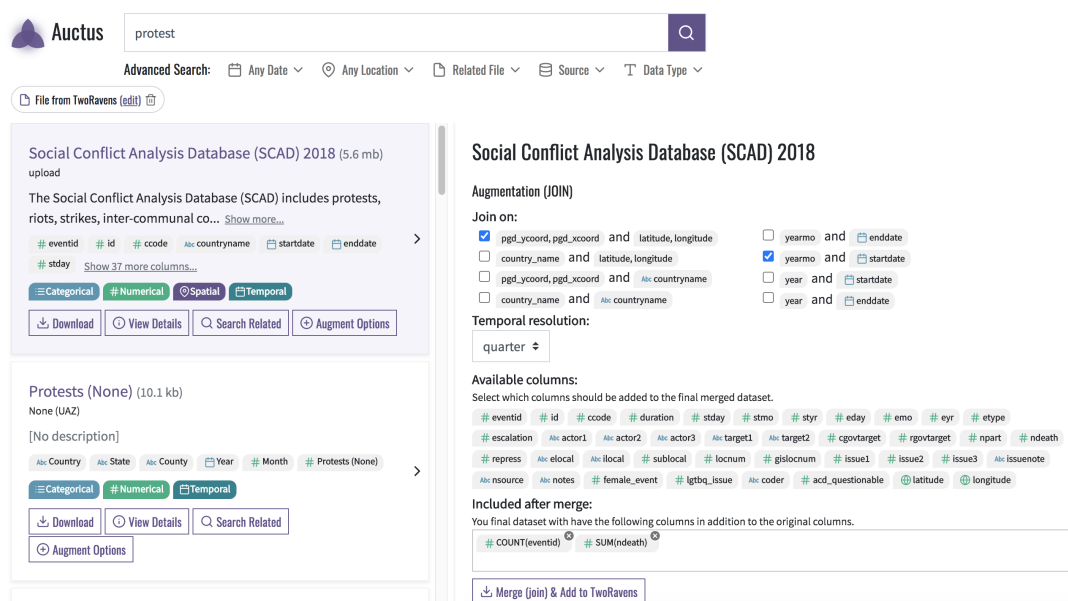


Figure 18: Auctus: The search results, that match the keyword ‘protest’ and the grid dataset, are displayed. The SCAD dataset is selected and the augmentation is set up to perform a spatial-temporal join.

Data exploration and model search specification. Initially, the expert loads a dataset about traffic collisions and taxi trips. Her goal is to create a model that predicts the number of collisions; such a model can allow her to explore what-if scenarios. After loading her dataset, Visus profiles it and displays visualizations that summarize the data. She will see a screen similar to Figure 6(C) which displays the different features, namely the *date*, *number of trips*, and *the number of collisions*, and how their values are distributed. In this same screen, she can also select the *target* variable she wants to predict and the type of model she wants to build (e.g., classification, regression). After pressing the “Configure Search” button, she is presented another screen where she can specify additional parameters of the model search specification (e.g., the maximum running time, evaluation metrics). The next step (“Start Solution Search”) triggers the execution of the AutoML system and a new screen is shown to display the results (“Explore Solutions”). As soon as models are generated by the AutoML system, Visus displays them in the “Solutions Table”, shown in Figure 7, which displays the generated pipelines and summarizes their evaluation scores in a histogram. To get more insight into the behavior of the different solutions, she moves to the next step, “Explain Solutions”, which produces explanatory visualizations such as the confusion scatter plot as seen in Figure 19(1). Analyzing the confusion scatter plot she notices that, for the pipeline with the smallest error, all instances are being predicted with the same values (all are in the same position on the *x*-axis). Clearly, this model is not very useful. Thus, she takes a step back and returns to the “Define Problem” screen to refine the problem.

Augmenting the data to increase performance. Now she hypothesizes that the unsatisfactory performance may be a result of insufficient predictive power of her data attributes. She then clicks on the tab “Data Augmentation”, to search for additional data. Intuitively, weather information, such as bad weather conditions can potentially increase the number of collisions. She uses the search form to find datasets that have the keyword “weather” and that are related to her input data, i.e., that can be joined with it, and then selects a matching dataset. Because many collisions also involve cyclists, she also searches for datasets related to her input data using the keyword “citi bike”, hoping to find a dataset containing information about the number of cyclists in the streets. Because both weather and citi bike datasets contain temporal attributes, as has her original data, Visus is able to automatically perform a temporal join of these datasets. After doing the augmentations with weather and Citi Bike data, her data increased from 3 to 7 features. She now re-runs the solution search to generate solutions using the augmented data and finds that the best pipeline has now improved from 71.29 to 61 in terms of mean absolute error, a significant improvement.

After picking her new favorite pipelines on the “Explore Solution” screen, she inspects them in more detail.

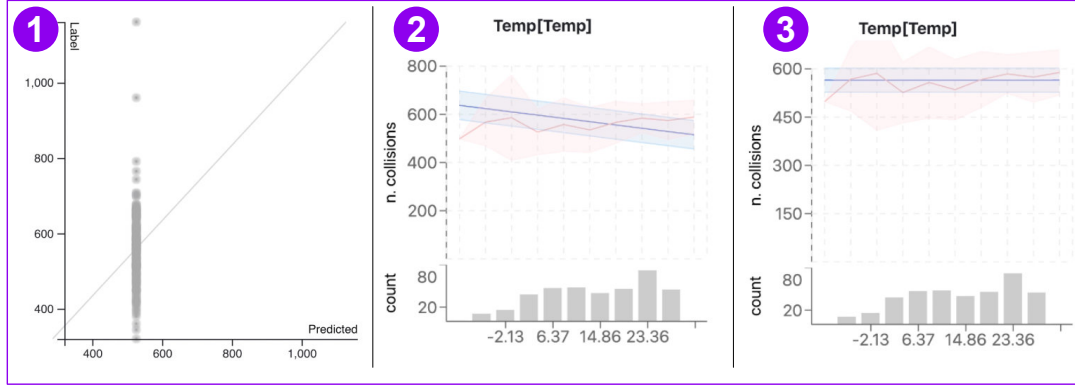


Figure 19: User-generated visualizations: (1) initial confusion scatter plot, (2) partial dependency plot for temperature from model A, (3) partial dependency plot for temperature from model B.

She notices that now her best pipeline’s predictions improved and the confusion scatter plot shows a more diagonal pattern (see Figure 8(F4)), which is the desired behavior. By using the partial dependency plot, she inspects the impact of the features in different models. She notices that, for pipeline A, the *temperature* has a strong negative correlation with the number of collisions (Figure 19(2)), while the pipeline B seems not to use temperature, as indicated by a flat line (Figure 19(3)) showing no correlation. Both pipelines use different versions of the Lasso algorithm and have similar performance in terms of error, she uses her domain knowledge, and decides to choose the model B which does not take temperature into account, even though its performance according to evaluation metrics is inferior when compared to A.

B. User Feedback

Visus’s design decisions have been grounded on feedback from domain-experts — with little or no data science knowledge — who interacted with earlier versions of our system. Here, we discuss the feedback received from users in more recent sessions. All users interacted with the system using data that came from their area of expertise and that they were already familiar with. For instance, one of the users has a Ph.D. in Biology who is interested in plant genetics and phenotyping and another is a manager at a U.S. agency who works with forward operators, soldiers, and analysts who want to analyze their data.

Users were generally satisfied with the system design, and were keen to start making connections to additional augmented data, wanted to overlay weather, add genetic information, and even geolocation. However, we noticed that there is still a gap in the communication of machine learning concepts to domain experts. For instance, it may not be clear for the users the meaning of evaluation metrics. *What is the difference between R^2 , mean absolute error, and mean squared error? When should one choose one over the other?*

Users also had some difficulty to understand and read the classification model explanations. For example, visual explanations like the Rule Matrix [86] summarize complex Bayesian classification rules derived from a surrogate model in a single visualization. In order to understand them properly, details had to be explained, including the meaning of a rule and statistical concepts such as fidelity and evidence. Although users claimed to understand the visualization after verbal explanations from an expert, it was not intuitive for them at first glance. This suggests that while such visualizations may be suitable for data scientists with machine learning knowledge, further research is necessary towards visual explanations geared towards non-expert audiences.

3.4.3 AlphaD3M

To demonstrate the effectiveness of AlphaD3M and its ability to handle a rich set of ML tasks, we compared AlphaD3M with state-of-the-art AutoML systems using two dataset collections.

A. D3M Datasets

This collection contains challenging datasets that go beyond tabular data and cover a wide variety of tasks (a total of 17 task types) and data types (see Table 4). We evaluated all the systems using train and test

Table 3: Comparison of MT-AutoML systems with respect to the number of supported task types, winner pipelines, and average rank by each system.

	AlphaD3M	Auto ⁿ ML	Ensemble	Aika	Distil	Autoflow	Axolotl
Unique ML tasks supported	17	16	15	17	15	16	14
Winner pipelines	49	39	30	21	20	11	10
Average rank	2.85	2.89	2.90	3.99	4.68	5.32	5.73

Table 4: Number of datasets by task type and number of solved datasets by each AutoML system for all task types covered by the D3M datasets.

ML Task	AlphaD3M	Auto ⁿ ML	Ensemble	Aika	Distil	Autoflow	Axolotl
Tabular Classification (20)	20	19	18	20	18	17	13
Tabular Regression (11)	11	11	11	8	9	6	5
Image Classification (9)	9	8	9	9	7	7	2
Image Regression (1)	1	1	1	1	1	1	1
Text Classification (9)	9	9	9	9	8	8	9
Audio Classification (2)	2	2	2	2	1	2	2
Graph Matching (3)	3	3	3	3	2	2	2
Time series Forecasting (13)	13	13	13	13	2	12	10
Link Prediction (3)	3	3	3	3	2	2	2
Collaborative Filtering (1)	1	0	1	1	0	1	0
Time series Classification (19)	19	19	19	17	19	15	19
Community Detection (3)	3	3	0	2	2	1	0
Video Classification (2)	2	2	2	2	0	2	2
Vertex Classification (4)	4	4	4	4	4	4	4
Object Detection (2)	2	2	0	1	1	0	0
Semisupervised Classification (6)	6	6	6	3	6	4	3
LUPI (4)	4	4	4	4	4	4	4

splits (see the dataset’s repository ¹⁴ for details). For each dataset, we ran the systems over the train split for one hour, a time-bound used by others works [51, 57]. After that, we evaluated the best pipeline produced by each system in the test split.

Table 3 shows the number of supported task types (ML tasks), winner pipelines (i.e., pipelines with the best performance for a given dataset), and the average rank by each AutoML system (rank of each system among the AutoML systems applied to each dataset). As we can see, AlphaD3M and Aika were able to solve 17 out of 17 unique tasks, obtaining the best coverage. We also evaluated the effectiveness of AlphaD3M. It had the best overall performance, producing the best pipeline for 49 datasets with the best average rank (2.85). Analyzing the support for each task type individually in Table 4, we can see that AlphaD3M was able to produce valid pipelines for all the datasets and it solved more datasets than the other systems.

B. OpenML Benchmark

Similar to [51], we compared our system with AutoWEKA, TPOT, H2O, AutoGluon, and Auto-Sklearn 2.0 (hereinafter referred to as Auto-Sklearn) on the 39 OpenML datasets [63]. This corpus contains a variety of datasets intended to represent real-world data science problems and covers binary and multiclass classification tasks. We used the AutoML Benchmark, AMLB [62] to compare the systems, running them locally for one hour using 1 fold split and accuracy as the optimization metric.

Figure 20 shows the scores (normalized by the best score) of all the systems. As we can see, AlphaD3M produced pipelines whose performance is on par with the other AutoML systems. We also calculated the average rank for all the systems for the 39 datasets. AlphaD3M got 3.64 of average rank, while Auto-Sklearn, AutoGluon, H2O, TPOT, and AutoWEKA got 2.08, 2.33, 3.08, 3.72, and 5.10, respectively. To understand better these numbers, we also estimated the performance gain of the pipelines found by AlphaD3M against pipelines generated by other systems. The average gain of AlphaD3M for the OpenML datasets was +0.001, which shows that, in general, AlphaD3M attained good results for this collection. We analyzed the 3 datasets (`task_146195`, `task_167119` and `task_168331`) for which AlphaD3M generated pipelines with performance lower than other systems. This happened because these datasets are imbalanced with multiple classes. The performance of AlphaD3M for these could be improved with the inclusion of primitives to handle imbalanced

¹⁴<https://datasets.datadrivendiscovery.org/d3m/datasets>

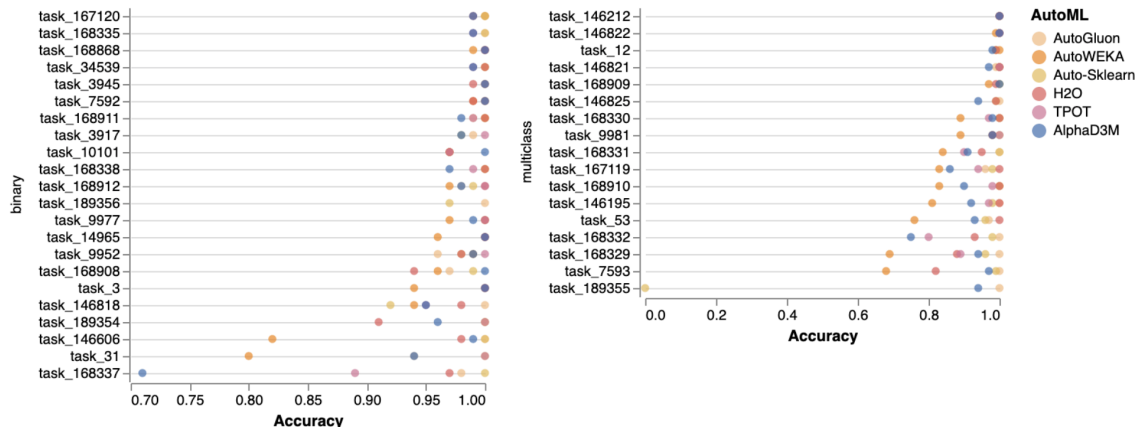


Figure 20: Performance of AutoML systems in OpenML Benchmark. X-axis shows the accuracy values (normalized by the best score), and Y-axis shows the IDs of the OpenML tasks.

datasets. This underscores the importance of being able to add primitives to AutoML systems.

Concerning the coverage, it is important to highlight that AlphaD3M succeeded for 38 datasets. Auto-Sklearn, AutoGluon, H2O, TPOT, and AutoWEKA solved 39, 39, 34, 29, and 28 datasets, respectively. As pointed out by [62], the results of Auto-Sklearn on the OpenML datasets must be considered very carefully, since there could be an overlap between the datasets used in its meta-learning process and the ones used in the evaluation. It’s important to highlight that none of the OpenML datasets are included in the version of Marvin that was used by AlphaD3M in these experiments.

3.4.4 Alpha-AutoML: JIATF Applications

We successfully participated in the data challenge organized by EOP (External Operations Platform) from JIATF. Together with Jataware Corp (another D3M team), our primary goal in the data challenge was to integrate D3M automated machine learning technology into the partner systems, along with supplementary tools that facilitate rapid data exploration/triage/annotation.

We developed a suite of interactive tools to streamline the complete ML model-building process, from data labeling to model training. Specifically, we developed three tools to facilitate the training of machine learning models to automatically classify images and documents of interest: (1) image triage app, (2) text triage app, and (3) Alpha-AutoML app. They enable analysts to quickly build reusable models for specific tasks, e.g., finding images containing weapons or finding documents containing written propaganda. All tools are Streamlit applications [117] that can be readily integrated into existing partner infrastructure.

We built the Alpha-AutoML Streamlit app as a UI wrapper around the open-source alpha-automl library (see Figure 21). It takes as input the training data (e.g., a CSV file) and automatically creates and evaluates a series of ML pipelines. Next, the user can select the desired model and download it for later usage. There are two main ways to deploy the downloaded model: the user can (1) develop custom Python code to integrate model prediction in any data processing pipeline or (2) use our built-in Streamlit app to compute predictions for data uploaded via a CSV or ZIP file.

As a use case, we considered an analyst who is trying to categorize different images obtained from a mobile phone. An image can be classified as relevant (when it is related to the usage of guns, terrorist activities, etc) or irrelevant (otherwise). To create the classification models, we ran Alpha-AutoML for 10 minutes. The system synthesized high-quality pipelines, with accuracy values around 0.91. Figure 22, shows some examples classified by the best ML model generated by Alpha-AutoML. As we can see, the first two images are classified as relevant, since they are related to terrorist activities. However, the last two images are classified as irrelevant because they are not related to this topic.

As a result of our participation in the data challenge, we obtained second-place awards in two categories “Best New Insights” and “Best User Interface Improvement”. We also got third-place awards in “Best Cross Application Integration” and “Grand Champions Winner” categories. Finally, we integrated our Streamlit app into the EOP system in the target network.

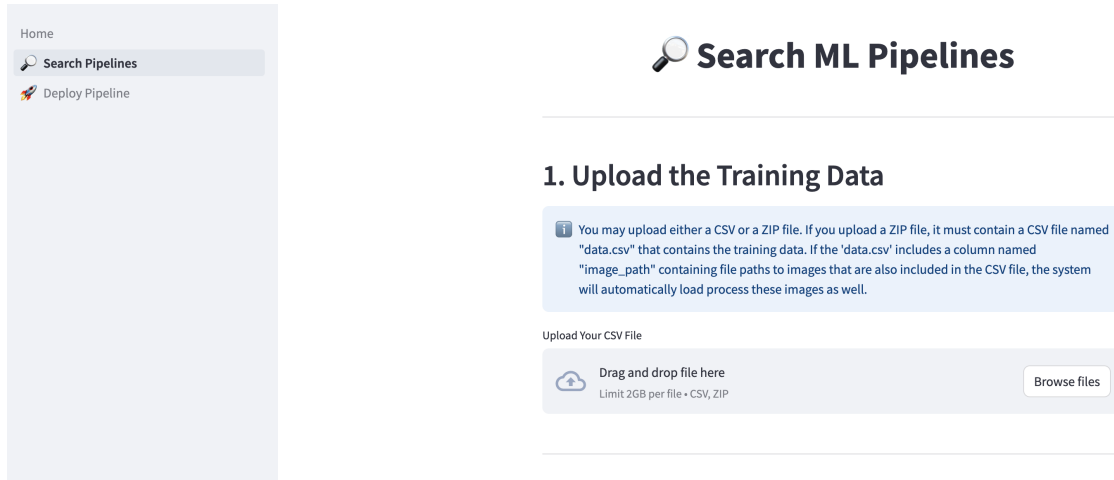


Figure 21: The Alpha-AutoML app allows users to upload the data created using the text and image labeling tools and allows users to build machine learning pipelines using the Alpha-AutoML tool.

4 CONCLUSIONS

In this project, we developed new methods and tools that support automatic machine learning, addressing three main challenges: (1) automatic and efficient synthesis of end-to-end machine learning pipelines for a wide range of tasks; (2) enabling domain experts to specify tasks as well as explore and assess the derived pipelines; and (3) support for data discovery to augment training data and improve models. For the first challenge, we developed AlphaD3M and Alpha-AutoML, open-source Python libraries that support multiple ML problems. To address the second challenge, we have designed Visus and PipelineProfiler, visual interactive tools for the exploration of pipeline collections derived by AutoML systems. For the third challenge, we implemented Auctus, an open-source dataset search engine that was designed to support data discovery and augmentation. The results of our project have been disseminated as open-source tools and papers published in international venues.

5 REFERENCES

- [1] ACLED: The Armed Conflict Location & Event Data Project. Accessed: 2021-09-30.
- [2] T. Alrashed, D. Paparas, O. Benjelloun, Y. Sheng, and N. Noy. Dataset or not? a study on the veracity of semantic markup for dataset pages. In *20th International Semantic Web Conference (ISWC 2021)*, 2021.
- [3] 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 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 337–346. ACM, 2015.
- [4] The Auctus Open-Source Dataset Search Engine. <https://github.com/VIDA-NYU/auctus>.
- [5] 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.
- [6] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In J. D. P. Rolim and S. Vadhan, editors, *Randomization and Approximation Techniques in Computer Science*, pages 1–10, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.
- [7] L. Barbosa and J. Freire. An adaptive crawler for locating hidden-web entry points. In *Proceedings of the 16th international conference on World Wide Web*, pages 441–450, 2007.

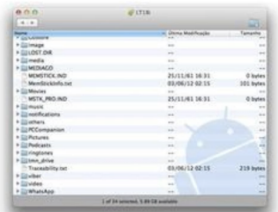
/3d/9gjmns916b30lplrfg_k62zr0000gn /T/tmp0p1oxfb5/terrorist_fighters /th-2136966618.jpeg	relevant	
/var/folders /3d/9gjmns916b30lplrfg_k62zr0000gn /T/tmp0p1oxfb5/terrorist_fighters /th-3922144560.jpeg	relevant	
/var/folders /3d/9gjmns916b30lplrfg_k62zr0000gn /T/tmp0p1oxfb5/mobile_android_files /th-2495754288.jpeg	irrelevant	
/var/folders /3d/9gjmns916b30lplrfg_k62zr0000gn /T/tmp0p1oxfb5/arab_memes /arab_memes100.jpeg	irrelevant	

Figure 22: Examples of images classified by the best ML model generated by Alpha-AutoML

- [8] L. Barbosa, K. Pham, C. Silva, M. Vieira, and J. Freire. Structured Open Urban Data: Understanding the Landscape. *Big Data*, 2(3), 2014.
- [9] O. Benjelloun, S. Chen, and N. Noy, editors. *Google Dataset Search by the Numbers*, 2020.
- [10] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for Hyper-Parameter Optimization. In *Proceedings of NIPS*, pages 2546–2554, 2011.
- [11] J. Bergstra and Y. Bengio. Random Search for Hyper-parameter Optimization. *JMLR*, pages 281–305, Feb. 2012.
- [12] K. J. Berry and P. W. Mielke Jr. A monte carlo investigation of the fisher z transformation for normal and nonnormal distributions. *Psychological Reports*, 87(3-suppl):1101–1114, 2000.
- [13] K. Beyer, R. Gemulla, P. J. Haas, B. Reinwald, and Y. Sismanis. Distinct-value synopses for multiset operations. *Commun. ACM*, 52(10):87–95, Oct. 2009.
- [14] K. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla. On synopses for distinct-value estimation under multiset operations. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 199–210, New York, NY, USA, 2007. ACM.
- [15] A. J. Bishara and J. B. Hittner. Confidence intervals for correlations when data are not normal. *Behavior Research Methods*, 49(1):294–309, 2017.
- [16] A. J. Bishara, J. Li, and T. Nash. Asymptotic confidence intervals for the pearson correlation via skewness and kurtosis. *British Journal of Mathematical and Statistical Psychology*, 71(1):167–185, 2018.

- [17] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 2011.
- [18] D. Brickley, M. Burgess, and N. Noy. Google dataset search: Building a search engine for datasets in an open web ecosystem. In *The World Wide Web Conference*, pages 1365–1375, 2019.
- [19] J. Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.
- [20] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, 1(1):538–549, 2008.
- [21] 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*, abs/1809.10782, 2018.
- [22] S. Castelo, R. Rampin, A. S. R. Santos, A. Bessa, F. Chirigati, and J. Freire. Auctus: A dataset search engine for data discovery and augmentation. *Proc. VLDB Endow.*, 14(12):2791–2794, 2021.
- [23] R. Castro Fernandez, J. Min, D. Nava, and S. Madden. Lazo: A cardinality-based method for coupled estimation of jaccard similarity and containment. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1190–1201, April 2019.
- [24] S. Chakrabarti. Focused web crawling. In L. Liu and M. T. Özsu, editors, *Encyclopedia of Database Systems*, pages 1147–1155. Springer US, 2009.
- [25] S. Chakrabarti, D. A. Gibson, and K. S. McCurley. Surfing the web backwards. *Computer Networks*, 31(11-16):1679–1693, 1999.
- [26] S. Chakrabarti, M. Van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer networks*, 31(11-16):1623–1640, 1999.
- [27] A. Chapman, E. Simperl, L. Koesten, G. Konstantinidis, L. D. Ibáñez-Gonzalez, E. Kacprzak, and P. T. Groth. Dataset Search: A Survey. *CoRR*, abs/1901.00735, 2019.
- [28] E. Chenoweth and J. Ulfelder. Can structural conditions explain the onset of nonviolent uprisings? *Journal of Conflict Resolution*, 61(2):298–324, 2017.
- [29] B. Cobarrubia. Dagre: Graph layout for javascript. <https://github.com/dagrejs/dagre/wiki>, 2018.
- [30] L. Columbus. IBM Predicts Demand For Data Scientists Will Soar 28% By 2020. <https://www.forbes.com/sites/louiscolumbus/2017/05/13/ibm-predicts-demand-for-data-scientists-will-soar-28-by-2020/>.
- [31] K. A. Cook and J. J. Thomas. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. National Visualization and Analytics Ctr, 5 2005.
- [32] G. Cormode, M. N. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1-3):1–294, 2012.
- [33] D3M Website. <https://datadrivendiscovery.org>, 2022.
- [34] D3M Primitives Website. <https://gitlab.com/datadrivendiscovery/primitives/-/tree/master/primitives>, 2022.
- [35] A. Dasgupta, J. Poco, Y. Wei, R. B. Cook, E. Bertini, and C. T. Silva. Bridging theory with practice: An exploratory study of visualization use and design for climate model comparison. *IEEE Trans. Vis. Comput. Graph.*, 21(9):996–1014, 2015.
- [36] Data.gov: U.S. Government’s open data. <https://www.data.gov/>.

- [37] Datamart Profiler Website. <https://pypi.org/project/datamart-profiler/>, 2021.
- [38] Dataverse. <https://dataverse.harvard.edu/>, 2021.
- [39] G. T. de Assis, A. H. Laender, A. S. da Silva, and M. A. Gonçalves. The impact of term selection in genre-aware focused crawling. In *Proceedings of the 2008 ACM symposium on Applied Computing*, pages 1158–1163, 2008.
- [40] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, M. Gori, et al. Focused crawling using context graphs. In *VLDB*, pages 527–534, 2000.
- [41] Docker. <https://www.docker.com/>, 2021.
- [42] N. Dolatnia, A. Fern, and X. Fern. Bayesian Optimization with Resource Constraints and Production. In *Proceedings of ICAPS*, pages 115–123, 2016.
- [43] H. Dong and F. K. Hussain. Self-adaptive semantic focused crawler for mining services information discovery. *IEEE Transactions on Industrial Informatics*, 10(2):1616–1626, 2012.
- [44] H. Doraiswamy, N. Ferreira, T. Damoulas, J. Freire, and C. T. Silva. Using Topological Analysis to Support Event-Guided Exploration in Urban Data. *IEEE transactions on visualization and computer graphics*, 20(12):2634–2643, 2014.
- [45] V. D’Orazio. Conflict Forecasting and Prediction. In *Oxford Research Encyclopedia of International Studies*. Oxford University Press, 2020.
- [46] V. D’Orazio, M. Deng, and M. Shoemate. Tworavens for event data. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 394–401. IEEE, 2018.
- [47] I. Drori, Y. Krishnamurthy, R. Lourenco, R. Rampin, K. Cho, C. Silva, and J. Freire. Automatic Machine Learning by Pipeline Synthesis using Model-based Reinforcement Learning and a Grammar. In *6th ICML Workshop on Automated Machine Learning*, 2019.
- [48] I. Drori, Y. Krishnamurthy, R. Rampin, R. de Paula Lourenco, J. Piazzentin Ono, K. Cho, C. Silva, and J. Freire. Alphad3m: Machine learning pipeline synthesis. In *5th ICML Workshop on Automated Machine Learning*, 2018.
- [49] Elasticsearch. <https://www.elastic.co/>, 2021.
- [50] J. Elliott. DARPA Data-Driven Discovery of Models (D3M) Program. <https://www.darpa.mil/program/data-driven-discovery-of-models>, 2020.
- [51] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data, 2020.
- [52] FAO FAOSTAT Data - Food and Agriculture Organization. <http://www.fao.org/faostat/en/#data>.
- [53] M. M. Farag, S. Lee, and E. A. Fox. Focused crawler for events. *International Journal on Digital Libraries*, 19(1):3–19, 2018.
- [54] A. Fedosejev. *React. js essentials*. Packt Publishing Ltd, 2015.
- [55] R. C. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker. Aurum: A Data Discovery System. In *ICDE ’18*, pages 1001–1012, 2018.
- [56] R. C. Fernandez, J. Min, D. Nava, and S. Madden. Lazo: A cardinality-based method for coupled estimation of jaccard similarity and containment. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1190–1201. IEEE, 2019.
- [57] M. Feurer, K. Eggenberger, S. Falkner, M. Lindauer, and F. Hutter. Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning, 2021.

- [58] M. Feurer, A. Klein, K. Eggenberger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In *Advances in neural information processing systems*, pages 2962–2970, 2015.
- [59] M. Feurer, A. Klein, K. Eggenberger, J. T. Springenberg, M. Blum, and F. Hutter. Auto-sklearn: Efficient and Robust Automated Machine Learning. In F. Hutter, L. Kotthoff, and J. Vanschoren, editors, *Automated Machine Learning: Methods, Systems, Challenges*, pages 113–134. Springer International Publishing, Cham, 2019.
- [60] J. Freire, A. Bessa, F. Chirigati, H. Vo, and K. Zhao. Exploring What not to Clean in Urban Data: A Study Using New York City Taxi Trips. *IEEE Data Engineering Bulletin*, 39(2):63–77, 2016.
- [61] J. Freire and C. Musco. Iii: Medium: Dataset search and ranking for data augmentation and explanation. Award # 2106888; Amount: \$1,093,195.00.
- [62] P. Gijsbers, M. L. P. Bueno, S. Coors, E. LeDell, S. Poirier, J. Thomas, B. Bischl, and J. Vanschoren. Amlb: an automl benchmark, 2022.
- [63] P. Gijsbers, E. LeDell, S. Poirier, J. Thomas, B. Bischl, and J. Vanschoren. An Open Source AutoML Benchmark. In *6th ICML Workshop on Automated Machine Learning*, 2019.
- [64] Y. Gil, J. Honaker, S. Gupta, Y. Ma, V. D’Orazio, D. Garijo, S. Gadewar, Q. Yang, and N. Jahan-shad. Towards human-guided machine learning. In *Proceedings of the 24th International Conference on Intelligent User Interfaces, IUI ’19*, pages 614–624, New York, NY, USA, 2019. ACM.
- [65] Y. Gil, K.-T. Yao, V. Ratnakar, D. Garijo, G. V. Steeg, R. Brekelmans, M. Kejriwal, F. Luo, and I.-D. Huang. P4ml: A phased performance-based pipeline planner for automated machine learning. In *Proceedings of Machine Learning Research, ICML 2018 AutoML Workshop*, 2018.
- [66] Google Cloud AutoML Website. <https://cloud.google.com/automl>, 2020.
- [67] M. Grover. Amundsen — Lyft’s data discovery & metadata engine. <https://eng.lyft.com/amundsen-lyfts-data-discovery-metadata-engine-62d27254fbb9>, 2019. [Online; accessed 20-October-2019].
- [68] 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.
- [69] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.
- [70] M. Hardt, E. Price, , and N. Srebro. Equality of opportunity in supervised learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3315–3323. Curran Associates, Inc., 2016.
- [71] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009.
- [72] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.
- [73] S. R. Hong, J. Hullman, and E. Bertini. Human factors in model interpretability: Industry practices, challenges, and needs. *Proceedings of the ACM on Human-Computer Interaction*, 4(CSCW1):1–26, 2020.
- [74] D. Huang, D. Y. Yoon, S. Pettie, and B. Mozafari. Joins on samples: A theoretical guide for practitioners. *arXiv preprint arXiv:1912.03443*, 2019.
- [75] M. Klein, L. Balakireva, and H. Van de Sompel. Focused crawl of web archives to build event collections. In *Proceedings of the 10th ACM Conference on Web Science*, pages 333–342, 2018.

- [76] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown. Auto-WEKA 2.0: Automatic Model Selection and Hyperparameter Optimization in WEKA. *The Journal of Machine Learning Research*, 18(1), 2017.
- [77] E. LeDell and S. Poirier. H2O AutoML: Scalable Automatic Machine Learning. *7th ICML Workshop on Automated Machine Learning (AutoML)*, July 2020.
- [78] O. Lehmberg, D. Ritze, R. Meusel, and C. Bizer. A large public corpus of web tables containing time and context metadata. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 75–76, 2016.
- [79] F.-F. Li and J. Li. Cloud AutoML: Making AI accessible to every business. <https://www.blog.google/products/google-cloud/cloud-automl-making-ai-accessible-every-business/>.
- [80] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.
- [81] R. Lopez, R. Lourenco, R. Rampin, S. Castelo, A. S. Santos, J. H. P. Ono, C. Silva, and J. Freire. Alphad3m: An open-source automl library for multiple ml tasks. In *AutoML Conference 2023 (ABCD Track)*, 2023.
- [82] Marvin. Marvin Website. <https://datadrivendiscovery.org/marvin>, 2020.
- [83] F. Menczer, G. Pant, and P. Srinivasan. Topical web crawlers: Evaluating adaptive algorithms. *ACM Transactions on Internet Technology (TOIT)*, 4(4):378–419, 2004.
- [84] Microsoft. Microsoft Azure Machine Learning Studio. <https://studio.azureml.net/>.
- [85] M. Milutinovic, B. Schoenfeld, D. Martinez-Garcia, S. Ray, S. Shah, and D. Yan. On evaluation of automl systems. In *Proceedings of the ICML Workshop on Automatic Machine Learning*, 2020.
- [86] Y. Ming, H. Qu, and E. Bertini. Rulematrix: Visualizing and understanding classifiers with rules. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):342–352, Jan 2019.
- [87] Nature Scientific Data. <https://www.nature.com/sdata/>.
- [88] Nyc vision zero initiative. <http://www1.nyc.gov/site/visionzero/index.page>.
- [89] National Oceanic and Atmospheric Administration. <https://www.noaa.gov/>.
- [90] Mayor de blasio signs new law lowering new york city’s default speed limit to 25 mph, October 2014. <http://tinyurl.com/zz35uur>.
- [91] NYC Vision Zero. <https://www1.nyc.gov/site/visionzero/index.page>.
- [92] NYC OpenData. <https://opendata.cityofnewyork.us>.
- [93] R. S. Olson, N. Bartley, R. J. Urbanowicz, and J. H. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016, GECCO ’16*, pages 485–492, New York, NY, USA, 2016. ACM.
- [94] R. S. Olson and J. H. Moore. TPOT: A Tree-based Pipeline Optimization Tool for Automating Machine Learning. In *ICML AutoML Workshop*, pages 66–74, 2016.
- [95] R. S. Olson, R. J. Urbanowicz, P. C. Andrews, N. A. Lavender, L. C. Kidd, and J. H. Moore. Automating biomedical data science through tree-based pipeline optimization. In G. Squillero and P. Burelli, editors, *Proceedings of the 18th European Conference on the Applications of Evolutionary and Bio-inspired Computation*, Lecture Notes in Computer Science, Berlin, Germany, 2016. Springer-Verlag.

- [96] J. P. Ono, S. Castelo, R. López, E. Bertini, J. Freire, and C. T. Silva. PipelineProfiler: A Visual Analytics Tool for the Exploration of AutoML Pipelines. *IEEE Transactions on Visualization and Computer Graphics*, 27:390–400, 2021.
- [97] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [98] J. Pei. A survey on data pricing: from economics to data science. *IEEE Transactions on Knowledge and Data Engineering*, 2020.
- [99] L. L. Pipino, Y. W. Lee, and R. Y. Wang. Data quality assessment. *Communications of the ACM*, 45(4):211–218, 2002.
- [100] J. Poco, H. Doraiswamy, H. T. Vo, J. L. D. Comba, J. Freire, and C. T. Silva. Exploring traffic dynamics in urban environments using vector-valued functions. *Computer Graphics Forum*, 34(3):161–170, 2015.
- [101] <https://www.prio.org/Data/PRI0-GRID/>, 2021.
- [102] U. N. W. F. Programme. *Ethiopia country brief*, 2020 (accessed May 2, 2021).
- [103] D. Qiu, L. Barbosa, X. L. Dong, Y. Shen, and D. Srivastava. Dexter: large-scale discovery and extraction of product specifications on the web. *Proceedings of the VLDB Endowment*, 8(13):2194–2205, 2015.
- [104] C. R. Rao. *Linear Statistical Inference and Its Applications*. Wiley, New York, 1973.
- [105] M. T. Ribeiro, S. Singh, and C. Guestrin. “why should i trust you?” explaining the predictions of any classifier. In *ACM KDD*, 2016.
- [106] S. Salehi and A. Schmeink. Data-centric green artificial intelligence: A survey. *IEEE Transactions on Artificial Intelligence*, 1(01):1–18, sep 5555.
- [107] I. Salehyan, C. S. Hendrix, J. Hamner, C. Case, C. Linebarger, E. Stull, and J. Williams. Social conflict in africa: A new database. *International Interactions*, 38(4):503–511, 2012.
- [108] A. Santos, A. Bessa, F. Chirigati, C. Musco, and J. Freire. Correlation sketches for approximate join-correlation queries. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1531–1544, 2021.
- [109] A. Santos, A. Bessa, C. Musco, and J. Freire. A sketch-based index for correlated dataset search. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 2928–2941, 2022.
- [110] 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)*, pages 1–7. Association for Computing Machinery, 2019.
- [111] D. J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. crc Press, 2003.
- [112] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *Conference on Neural Information Processing Systems*, 2017.
- [113] S. Sizov, M. Theobald, S. Siersdorfer, G. Weikum, J. Graupmann, M. Biwer, and P. Zimmer. The bingo! system for information portal generation and expert web search. In *CIDR*, 2003.
- [114] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

- [115] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. M. A. Patwary, P. Prabhat, and R. P. Adams. Scalable Bayesian Optimization Using Deep Neural Networks. In *Proceedings of the ICML*, pages 2171–2180, 2015.
- [116] The Socrata Open Data API. <https://dev.socrata.com>.
- [117] A faster way to build and share data apps. <https://streamlit.io/>, 2020.
- [118] R. Sundberg and E. Melander. Introducing the ucdp georeferenced event dataset. *Journal of Peace Research*, 50(4):523–532, 2013.
- [119] D. M. Team. D3m metalearning database. <https://metalearning.datadrivendiscovery.org/>, 2020.
- [120] A. F. Tollefsen, H. Strand, and H. Buhaug. Prio-grid: A unified spatial data structure. *Journal of Peace Research*, 49(2):363–374, 2012.
- [121] R. Turner, D. Eriksson, M. McCourt, J. Kiili, E. Laaksonen, Z. Xu, and I. Guyon. Bayesian Optimization is Superior to Random Search for Machine Learning Hyperparameter Tuning: Analysis of the Black-Box Optimization Challenge 2020. *CoRR*, abs/2104.10201, 2021.
- [122] M. L. Vidal, A. S. da Silva, E. S. de Moura, and J. M. Cavalcanti. Structure-driven crawler generation by example. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 292–299, 2006.
- [123] 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.
- [124] R. Y. Wang and D. M. Strong. Beyond accuracy: What data quality means to data consumers. *Journal of management information systems*, 12(4):5–33, 1996.
- [125] WHO: The World Health Organization. <https://www.who.int/data/collections>.
- [126] G. T. Wilson. Time Series Analysis: Forecasting and Control, 5th Edition. *Journal of Time Series Analysis*, 37(5):709–711, September 2016.
- [127] M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Learning Hyperparameter Optimization Initializations. In *2015 IEEE international conference on data science and advanced analytics (DSAA)*, pages 1–10. IEEE, 2015.
- [128] World Bank Open Data. <https://data.worldbank.org/>, 2021.
- [129] World Bank Open Data. <https://data.worldbank.org>.
- [130] Y. Yang, Y. Zhang, W. Zhang, and Z. Huang. Gb-kmv: An augmented kmv sketch for approximate containment similarity search. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 458–469, April 2019.
- [131] H. Yu and M. Zhang. Data pricing strategy based on data quality. *Computers & Industrial Engineering*, 112:1–10, 2017.
- [132] M. B. Zafar, I. Valera, M. Gomez Rodriguez, and K. P. Gummadi. Fairness beyond disparate treatment & disparate impact: Learning classification without disparate mistreatment. In *Proceedings of the 26th International Conference on World Wide Web, WWW '17*, pages 1171–1180, Republic and Canton of Geneva, Switzerland, 2017. International World Wide Web Conferences Steering Committee.
- [133] Zenodo. <https://zenodo.org/>.
- [134] H. Zhang, R. López, A. Santos, J. Piazzentin Ono, A. Bessa, and J. Freire. Using pipeline performance prediction to accelerate automl systems. In *Proceedings of the Seventh Workshop on Data Management for End-to-End Machine Learning*, pages 1–11, 2023.

- [135] H. Zhang, A. S. R. Santos, and J. Freire. DSDD: domain-specific dataset discovery on the web. In G. Demartini, G. Zuccon, J. S. Culpepper, Z. Huang, and H. Tong, editors, *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, pages 2527–2536. ACM, 2021.
- [136] Y. Zhang and Z. G. Ives. Finding related tables in data lakes for interactive data science. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1951–1966, 2020.
- [137] E. Zhu. *Find Open Data*, 2019 (accessed May 2, 2021).
- [138] E. Zhu, D. Deng, F. Nargesian, and R. J. Miller. Josie: Overlap set similarity search for finding joinable tables in data lakes. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD '19*, pages 847–864, New York, NY, USA, 2019. ACM.
- [139] E. Zhu, D. Deng, F. Nargesian, and R. J. Miller. Josie: Overlap set similarity search for finding joinable tables in data lakes. In *Proceedings of the 2019 International Conference on Management of Data*, pages 847–864, 2019.
- [140] E. Zhu, F. Nargesian, K. Q. Pu, and R. J. Miller. Lsh ensemble: Internet-scale domain search. *Proc. VLDB Endow.*, 9(12):1185–1196, Aug. 2016.
- [141] I. Zliobaite. A survey on measuring indirect discrimination in machine learning. *CoRR*, abs/1511.00148, 2015.