

# Visual Analytics for Automated Model Discovery

Dylan Cashman\*, Shah Rukh Humayoun\*, Florian Heimerl, Kendall Park, Subhajit Das, John Thompson, Bahador Saket, Abigail Mosca, John Stasko, Alex Endert, Mike Gleicher, Remco Chang

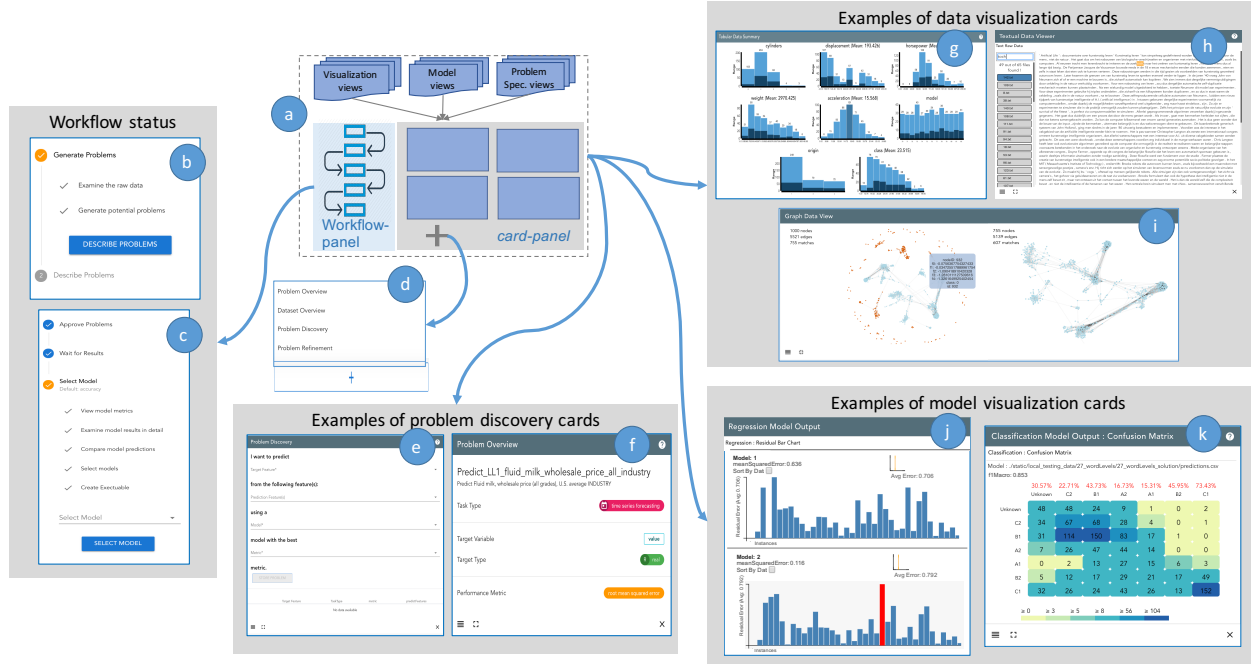


Fig. 1. The various components of our visual analytics system, Snowcat, that facilitates automated model discovery. (a) shows an abstract view of the system interface and workflow, (b) and (c) show the *workflow-panel* at different stages (e.g., problem specification, training models, model selection) of exploratory visual analytics with autoML, (e) and (f) are examples of problem discovery and problem specification cards, (g) – (i) are examples of data visualization cards for tabular, time-series and graph datasets, and (j) and (k) are examples of model visualization cards through residual bar chart and confusion matrix, and (d) shows a possible list of adding cards at some stage of workflow execution.

**Abstract**—A recent advancement in the machine learning community is the development of automated machine learning (autoML) systems, such as autoWeka or Google's Cloud AutoML, which automate the model selection and tuning process. However, while autoML tools give users access to arbitrarily complex models, they typically return those models with little context or explanation. Visual analytics can be helpful in giving a user of autoML insight into their data, and a more complete understanding of the models discovered by autoML, including differences between multiple models. In this work, we describe how visual analytics for automated model discovery differs from traditional visual analytics for machine learning. First, we propose an architecture based on an extension of existing visual analytics frameworks. Then we describe a prototype system Snowcat, developed according to the presented framework and architecture, that aids users in generating models for a diverse set of data and modeling tasks.

**Index Terms**—Visual analytics framework, machine learning models, automated machine learning (autoML), confirmatory data analysis, exploratory data analysis, visual analytics system design.

## 1 INTRODUCTION

Automated machine learning, or autoML, is a recent advancement from the machine learning (ML) community that holds the promise to

revolutionize the use of ML algorithms for data analysis [43]. Instead of requiring an analyst or data scientist to manually select an ML algorithm, tune the parameters, and validate its effectiveness (e.g., via cross-validation) as is the traditional practice, an autoML system only requires the user to provide a task and a set of training data. The autoML system automatically searches through a collection of ML algorithms and their respective hyperparameters, and returns the “best” model that fits the user’s task and data.

Automated ML tools have sparked interest due to their ability to bring complex modeling to users without requiring users to have in-depth knowledge of the inner creation or workings of the resulting models. Already, libraries such as AutoWeka [24, 43], Hyperopt [5, 23], and Cloud AutoML (from Google) [26] are becoming available either commercially or as open-source tools. However, autoML is not a

- \* indicates equal contribution.
- Florian Heimerl, Kendall Park, and Mike Gleicher are with the University of Wisconsin – Madison. emails: {heimerl, kendall, gleicher}@cs.wisc.edu
- Subhajit Das, John Thompson, Bahador Saket, Alex Endert, and John Stasko are with Georgia Tech. emails: {das, jrthompson, saket, endert}@gatech.edu, john.stasko@cc.gatech.edu
- Dylan Cashman, Shah Rukh Humayoun, Abigail Mosca, and Remco Chang are with Tufts University. emails: {dcashm01, humayoun, amosca01, remco}@cs.tufts.edu.

panacea for those with data science needs. AutoML tools can be difficult to prepare and run; they require a precisely labeled and curated dataset and a well-defined problem specification. They typically offer no support for interpreting the resulting models. For example, imagine a data analyst working at a hospital, Jessie, looking to build a model for predicting patient stay time. While she would traditionally use linear regression, she hopes to build a better model with the help of an autoML tool. The autoML tool may produce that model, but Jessie needs to be sure that it is making reasonable predictions. She also needs to be able to communicate information about the model to other stakeholders, and she feels she should be able to use her own domain expertise to steer the autoML tool in some way. In short, what Jessie needs is a complete visual analytics environment where she could perform data exploration, data analysis, and data-driven decision making.

For the field of visual analytics (VA), the availability of autoML holds great promises. AutoML can make VA systems more powerful by connecting a wide array of complex learning algorithms to the user's data analysis tasks. It frees up the user from tweaking the low-level details of ML algorithms and techniques. Instead, the user can focus on the high-level tasks and goals without having to make decisions about the model building process. The VA system can enable the user to interact with models returned by an autoML tool in order to mitigate issues of trust and understanding.

However, integrating autoML with VA also comes with great challenges. VA systems typically integrate tightly with their respective models, visualizing their hyperparameters to provide the user with insight into the model. In contrast, autoML is often a black-box system that performs complex optimizations over a variety of ML models with obtuse hyperparameters. Not only is it difficult for a VA system to be fully integrated with such a system to control its inner workings, but it is even more difficult for a user to interact with it in a human-in-the-loop fashion. Nothing about the resulting models can be known by the application a priori to the user uploading data and the autoML tool running: this forces the application designer to avoid integrating the behavior of the system tightly with any one learning algorithm, such as SVM or decision tree, since the user experience should be consistent no matter what models the autoML returns. The application must also have a very wide scope of functionality to support many different types of data (numeric, text, images, graphs) across a variety of problem types (classification, regression, collaborative filtering, community detection).

In this paper, we examine how a VA system can be integrated with autoML. We begin by reviewing the roles of VA and the use of ML algorithms in VA systems and explain how the addition of autoML fits within existing VA frameworks with a few extensions. Because autoML requires a well-specified formal problem specification as input, a VA system using autoML needs to delineate the roles of data exploration and problem generation (or hypothesis generation) into distinct but inter-connected roles. We describe implications on architecture based on the VA system connecting to an autoML backend. Based on this, we outline a proposed client-server architecture (Section 4).

Unlike most typical VA systems which directly integrate an ML algorithm within a specialized visual interface. Our architecture emphasizes the need for a middleware that acts as a controller between the data source, the autoML, and the client visualization. Maintaining and supporting the communication between these elements is challenging due to their diverse needs and natures. For example, autoML searching for a “best” machine learning model can be slow – in the order of minutes if not hours. This delay should not affect the interactivity of the front-end visualization client, which necessitates the use of asynchronous two-way communication channels, both between the middleware and autoML as well as between the middleware and the client visualization.

The second contribution of this paper is an implementation of a system, Snowcat, based on the proposed architecture (Section 6). Snowcat is developed as part of the DARPA D3M program [39]. It is based on web-technology, combining Vue.js and d3.js for the browser-based frontend, and node.js for the middleware. It interfaces with a variety of different autoML backends being developed as part of the D3M program. It supports a user exploring a dataset, coming up with well-

defined ML problems, generating a set of models to answer those problems, and selecting the model that best fits their needs. We demonstrate the usage of Snowcat through two case studies using D3M autoML systems (Section 7). We show that Snowcat is robust in that it can support diverse data types and ML task types. Due to its modular design, it can be extended to include new visualization designs for new data types or ML tasks. In short, we demonstrate that Snowcat can effectively integrate autoML to support users in a range of data exploration and data analysis tasks.

## 2 WHAT IS AUTOML?

AutoML comprises a set of techniques designed to automate the end-to-end process of ML. It supposes that, with a well-defined task, a desired outcome (metric), and training data, autoML should be able to produce an optimal ML model without further human involvement. To accomplish this, autoML techniques automate a range of ML operations, including but not limited to, data cleaning, data pre-processing, feature engineering, feature selection, algorithm selection and hyperparameter optimization [15]. The goal of autoML is to hide the complex manipulation of these processes from the user, allowing them to benefit from the use of advanced ml without expertise in statistics or ml.

Since autoML is a relatively recent advancement in the machine learning community, the exact role and function of an autoML system are still undefined. For example, some autoML engines include parts of the machine learning pipeline while others don't (e.g., automated data cleaning [48]). Similarly, the inputs and outputs of an autoML system have not been standardized. While all systems require a formal task specification (including metrics) and training data as input, the exact format of the specification is system-dependent.<sup>1</sup> Further, what autoML should produce as an output is also unclear. Beyond producing an “optimal” model and its performance metrics, some autoML systems may produce the top  $k$  models. It could also be valuable for an autoML system to allow further access into the modeling processes, such as the evaluations used to produce the performance metrics (e.g., the details of the internal cross-fold validations used during the model search). This information could be analyzed to provide insight into what parts of the data are hardest to model.

To automatically produce a “best” model, autoML needs to search through a large number of algorithms and their associated hyperparameters, potentially training each discovered model before sampling a new one. Posed as an optimization problem, the goal of autoML is therefore to maximize the user-specified outcome (or metric) quickly and efficiently. Successful autoML systems attempt to use a better-than-random sampling strategy to iteratively sample from the learning algorithm and hyperparameter spaces. For example, Auto-WEKA [24, 43] uses Bayesian optimization techniques to iteratively choose from a subset of the available learning algorithms and hyperparameters in the Java-based WEKA machine learning tool [16]. Hyperopt [5, 23] extends a similar technique by parallelizing the search through model space, and interfacing with the Python-based `scikit-learn` [34] machine learning library. Other autoML tools develop sophisticated prior beliefs on which learning algorithms and hyperparameters work best on which datasets by precomputing their performance on open source datasets [20, 41]. Then, for a new dataset, they can sample from the model space according to where similar datasets performed well.

While the goal of autoML is to hide the technical details of machine learning from the user, in practice these systems are still out of the reach of most non-technical users because the use of autoML requires significant knowledge about machine learning or data science. For example, selecting a task like “link prediction” presumes that the user has modeled their data and problem as a graph (see Table 1) and that the user is looking to evaluate the possible relationships between the nodes. Similarly, when performing a classification task, the user needs

<sup>1</sup>For the system described in this paper, we adopt the API standards of the D3M program, which are regularly updated at <https://gitlab.com/datadrivendiscovery>. It is reasonable to expect an autoML VA system to either work closely with their autoML backend developers, as is our case, or to develop their own adapters for existing tools such as autoWeka or CloudML.

to specify performance metrics such as area-under-the-curve (AUC), F1 score, uncertainty coefficients, etc. as input to autoML.

### 3 RELATED WORK AND MOTIVATION

VA approaches integrate interactive visual interfaces and data mining to help users analyze patterns in their data [12]. In this section, we examine how the introduction of autoML affects the nature of a VA system by first reviewing previous VA approaches that include model creation from users’ data. We then consider existing theoretical frameworks of information visualization and VA, and then motivate and synthesize our own framework geared towards the inclusion of autoML into VA approaches.

#### 3.1 General Visual Analytics Systems

Visual analytics approaches often integrate Machine learning methods to create models from data to help users discover and analyze patterns from data. Disfunction [7] allows the user to select a distance metric by clicking and dragging data points together or apart to represent similarity. Utopian [11] allows users to steer the creation of topic models to better understand collections of texts by providing feedback to the modeling algorithm through keywords and example documents. Explainers [13] learns new data dimensions that correspond to specific concepts, and lets users lay out their data according to them.

The previous approaches help users better understand their data through modeling. Another task that visual analytics systems support is to help users get a better understanding of the models themselves through their data with the goal of either helping users improve models or selecting the best models from a set of candidates. Baobabview [45] supports users with iteratively refining decision tree models to achieve a better fit to their dataset. Mühlbacher et al. [32] help users balance potentially conflicting objectives such as accuracy and interpretability of decision tree models by facilitating comparison of candidate tree models. Heimerl et al. [17] support the task of refining binary classifiers for document retrieval by letting users interactively modify the classifier’s decision on any document.

Instead of focusing on single models and their refinement, some approaches help users with the task of selecting the most suitable model from a set of candidates. Mühlbacher and Piringer [33] support analyzing and comparing regression models based on visualization of feature dependencies and model residuals. Podium [49] aids a user in selecting a ranking model. With squares [35], analysts can compare classification models based on an in-depth analysis of label distribution on a test data set.

Recently, visual analytics research has focused on deep learning models, which are very successful in a wide range of domains, but often complex and thus hard to analyze and understand. Liu et al. [28] improve the interpretability of neural networks, by creating visual representations that expose their structure through grouping neurons with similar semantic properties. Other approaches support analyzing and debugging the training process of deep generative models [27], provide an abstract overview of the structure of and dataflow in such models [51], or help to provide insight into neural sequence models [40].

All of the above approaches support a very specific model types, assuming that potential users have already decided in favor of a specific type of method to model their data. None of these systems is designed to help users review and filter the large space of potential model types that could be applied on a given data set, and help them make informed decisions about potential goals of the modeling process, as well as suitable learning algorithms for achieving those goals.

#### 3.2 Visualization and Visual Analytics Frameworks

Information Visualization seeks to generate meaningful visual representations of data (Figure 2(a)). Chi and Riedl [10] proposed the *InfoVis reference model* (later refined by Card, Mackinlay and Shneiderman [9]) that emphasizes the mapping of data elements to visual forms. The framework by van Wijk [46] extends this with interaction – a user can change the specification of the visualization to focus on a different aspect of the data.

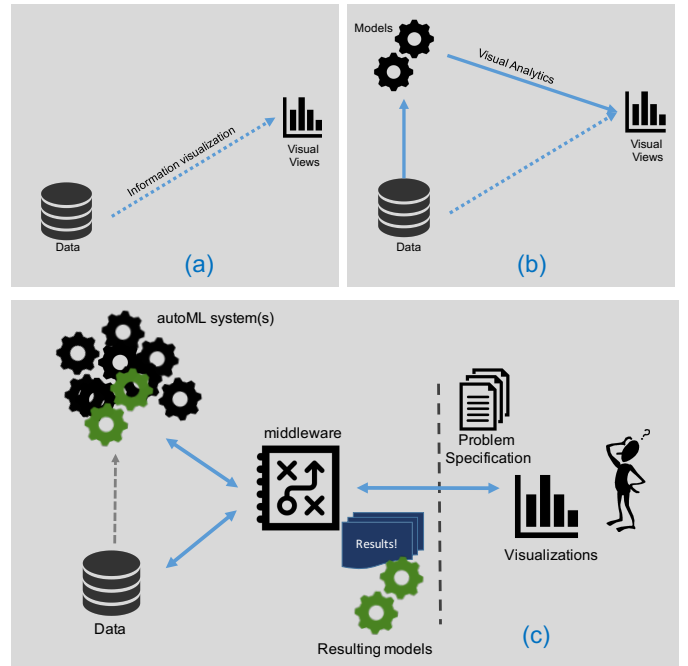


Fig. 2. (a) Information visualization maps data to visual elements. (b) In Visual Analytics, data-driven models are added to help interpret data. (c) Our proposed VA-autoML architecture. It incorporates a middleware component to facilitate the communication between the client visualization and the autoML systems.

In contrast, visual analytics makes use of **models** (Figure 2(b)). It emphasizes the interaction between the data, the models, and the visualization [22]. A model is defined as the outcome of a computational process, such as the use of a machine learning algorithm (Figure ??). Interacting with a visualization can trigger building a model or tuning its parameters. While Andrienko et al. [3] see the purpose of visual analysis in building models from data, a recent extension of Keim et al.’s framework describes how the interaction between models, visualizations, and users can generate new knowledge [38].

With autoML, which automates model building and parameter tuning, the role of VA needs to be reconsidered. Instead of interacting with the visualization to build and refine models, the user’s role can now be abstracted to a higher level of interacting with the “outcomes” of the models. Figure 2(c) illustrates this new relationship – instead of interacting directly with the raw data or the machine learning models, the user now interacts with the VA system to formalize a “problem specification”. In return, the VA system supports the user in understanding the models generated by autoML via visualizations of their outputs and performance metrics (e.g., accuracy, fitness, etc.).

#### 3.3 A VA-autoML Framework

The mechanisms and workflow of how analysts use a visual interface to interact with machine learning models have been examined previously. Wang et al. [50] extended the *models* phase in the framework by Keim et al. (green box in Figure 3(a)) to include a model-building process (Figure 3(b)) with: *feature selection and generation*, *model building and selection*, and *model validation*.

Andrienko et al. [3] proposes a similar, but more complex workflow of visual analytics. Compared to Keim et al. and Wang et al., their workflow assumes that the outcome of the VA process to either be an “answer” (to a user’s analysis question) or an “externalized model”. This perspective, where the outcome of visual analysis is a “best” model is akin to autoML. However, while autoML views model-building as a fully automated process, Andrienko et al. argue in favor of a human-in-the-loop analysis process, with a model as its final product.

Previous VA frameworks (Figure 3(a) and (b)) fail to capture the

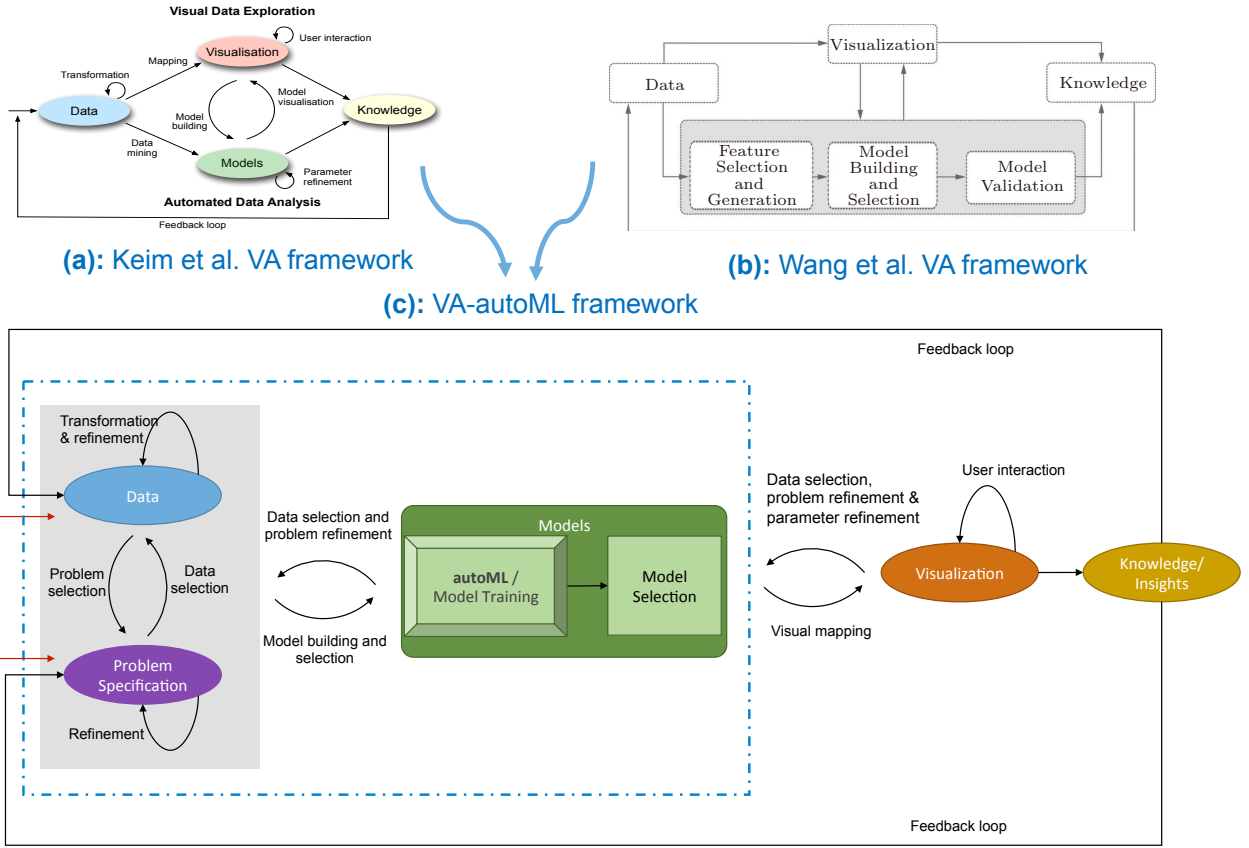


Fig. 3. Our extended VA-autoML framework based on the VA framework by Keim et al. [21, 22] (a) and Wang et al. [50] (b). Our framework adds a *Problem Specification* component and expands the *Models* component from Keim et al. to include the use of *autoML* for *Model Training* and the *Model Selection*. In addition, the framework distinguishes the exploratory data analysis (EDA) from confirmatory data analysis (CDA) as different analysis processes.

integration of autoML in a VA system for two main reasons. First, an autoML system is typically an opaque process and it is difficult to gain access to its internal workings. Instead, a VA system needs to rely on formal APIs to communicate with an autoML system. Second, most autoML systems require a formal problem specification as input. In previous VA frameworks and systems, *problem specifications* are not considered a relevant component, because VA is commonly considered an exploratory data analysis (EDA) tool [44]. In EDA settings, users are not expected to start analysis with a formal goal in mind.

However, if the goal of visual analysis is to generate “a best model” (e.g., in the sense of autoML and the workflow by Andrienko et al.), then it implies a well-specified problem for which a model can be quantitatively measured to be “best”. In this regard, VA can be considered a confirmatory data analysis (CDA) [19] tool as well. A user should thus be able to start the data analysis process with a specific goal and use the VA tool to create a model that best confirms (or refutes) the user’s initial hypothesis.

To capture these two additional considerations when using an autoML system, we propose a VA-autoML framework (Figure 3(c)). Here, VA is primarily used around the inputs and outputs of autoML (marked by the blue box), reflecting the black-box nature of autoML. In addition, a user could start the data analysis process with data exploration or problem specification. While both data and problem specification are required to start an autoML process, the two starting points reflect the differences between exploratory vs. confirmatory data analysis.

## 4 A VISUAL ANALYTICS ARCHITECTURE FOR AUTOML

Applying our VA-autoML framework, we propose a system architecture based on a typical client-server web architecture. AutoML, database, and the middleware reside on the server, and visualization on the web

client (Figure 2(c)). Communication between the components are shown as (directed) edges. The dashed line between the database and autoML represent direct access to the data source for efficiency purposes, but it is not a strict requirement for the functionality of the architecture.

### 4.1 Architecture Components

Our architecture has 4 software components and 3 communication channels (Figure 2(c)). The software components are: the autoML system, a database, a middleware, and a client. The communication channels comprise: autoML-middleware, database-middleware, and client-middleware. In the following, we describe the role and the design of each of the components.

#### 4.1.1 autoML

The autoML component is treated as a black box in our architecture. Its input is a problem specification and training and testing data. We assume that the autoML system returns the top  $k$  models for a given problem, as well as model predictions on a held out validation set. The autoML system can also respond to inquiries about its internal state. For example, when training a complex model, an autoML systems could respond with its status towards completing the task.

#### 4.1.2 Database

Data storage and retrieval can be challenging in a VA-autoML system, especially when the data is large. This complexity arises from the fact that the database is often not “read only”. Instead, both the client and the autoML system can write to the data. This can happen during the feature engineering phase, where the autoML system is adding new

(derived) attributes to the data, or when the user alters the data (through visualization) for data cleaning or augmentation.

The strategy for controlling this read-write database is more complicated when the data is large. Should the database allow changes to the raw data, make copies of the data every time it is changed, or record “diffs” that represent each modification to the data? Each of these strategies have benefits and drawbacks. While the design of the architecture is not immediately affected by the choice of the strategy, it may affect the choice of the communication protocol (see Section ??) between the database, the autoML system, and the middleware. For additional information about appropriate strategies for data versioning, we refer the readers to systems such as DataHub [6], VisTrails [8], and Apache Arrow [31].

#### 4.1.3 Client (Visualization)

Unlike most visualization systems that often only render data, a required capability of a visualization in our architecture is that it must be able to render “visualizations of models” [36,42]. While visualizations of raw data have been the core of visualization research, the design of visualizations of models is less understood. In Section 6.2, we describe some of these model visualizations that we have incorporated into Snowcat .

#### 4.1.4 Middleware

As an intermediary between autoML and the client, the middleware converts user interactions into formal, well-specified queries for the autoML system. Conversely, the middleware also needs to accept the outputs of autoML and relay them to the client using a predetermined API. In addition, when forwarding data between database and client (and database and autoML), the middleware’s job can include data sampling or selection. When the data is too large for the client, the middleware can automatically sample to reduce its size [4] or progressively stream the data to the client in an online fashion [18].

Our middleware can also connect to *multiple* autoML systems simultaneously. Some autoML systems favor particular “types” of models (e.g., Google’s Cloud AutoML system may produce deep neural networks [25] which are not part of the model search space for AutoWEKA [43]). Providing the VA system and the user with diverse models from different autoML systems encourages comparison that can lead to better model discovery. In addition, the middleware also integrates more traditional machine learning libraries (e.g., scikit-learn). This allows us to do some machine learning tasks, such as dimensionality reduction using t-SNE [29] for visualizing high-dimensional data, quicker and with more user control compared to using autoML for them.

### 4.2 Communications between Components

Our architecture has three communication channels between components: autoML-middleware, database-middleware, and client-middleware. Although similar to most client-server architectures in many aspects, there are also unique requirements when considering the integration of autoML. Below we describe the role of each of the communication channels with focus on their unique requirements.

**Client-middleware** communication has to be asynchronous, so that the client is not blocked while waiting for a server’s response. In addition, we require a two-way channel to support bi-directional communication. While the client frequently initiates requests, the middleware (in particular as a proxy for the autoML component) may also need to initiate requests to the client, e.g., for status updates or progressive streaming of results.

**AutoML-middleware** communication, by extension, needs to be bi-directional and asynchronous as well. In addition, the autoML API must be fully supported on this channel. While we are not aware of a standardized API for autoML, supported features must include problem specifications, information about task types and performance metrics, and the source data (including information about potential folds for model cross validation). Further useful features include the communication of user feedback and constraints to the autoML engine. The

autoML system must be able to communicate results with performance metrics for each model. It may also include richer feedback on the training process, including status updates and information about partial solutions.

**Database-middleware** communication needs to synchronize between the middleware and the autoML engine. In the simplest case, both components have read-only access to the database, but at least the middleware should write access (in case, e.g., the user wants to change, fix an error, or add to the data). In our current design, we allow arbitrary read-only access from both autoML and the middleware. Further changes to the data are restricted to the middleware, which acts as the sole owner of the database. This two-pronged approach assumes that update logic is handled by the middleware, but ensures that data access and start-up times (when the user loads a new dataset) are minimized.

### 5 IMPLEMENTATION CHALLENGES

Based on our architecture, we implemented a prototype VA system, **Snowcat** . The development of a *human-in-the-loop* VA system that supports autoML poses challenges that differ from traditional VA systems that support using a specific machine learning application, for example to perform confirmatory data analysis of binary classification over binary features [42]. Existing VA systems interact with the underlying models by making use of an integrative approach, exposing the parameters of the model to help the user collaboratively build their model. We instead desire a VA system that supports many types of tasks (e.g., classification, regression, graph matching, time-series forecasting, etc.) over many different types of data (e.g., tabular, graph, images, text, etc.) supported by underlying autoML engines. The design of a VA-autoML system must address the following challenges.

- **C1: Modular Design:** Our system must not be targeted towards specific modeling tasks, model or data types. Since the set of tasks, models, and data types to support may change at any time, the system must allow for modular extensions, so that components can be added dynamically.
- **C2: Asynchronous Execution:** Without *a priori* knowledge of the scale of the dataset, our system must gracefully handle datasets of millions of rows and thousands of features. The system must allow for workflows to be engaged with asynchronously so that modeling tasks of different time scales, such as the training of deep networks, would not block software execution.
- **C3: User Workflow Support:** To support the backtracking and feedback loop in the proposed VA-autoML framework, clear and afforded transitions in the resulting system are necessary. Users should be encouraged to pass back and forth between the data analysis steps (e.g., the modeling process, shown as the green boxes in Figure 3), while always being made aware of the next step to complete their task.
- **C4: Intuitive Interface:** Just as autoML supports many different types of models, data types, and domains, our system must be equally agnostic. As different domain experts, such as cyber security analysts or medical analysts, are familiar with different workflows and terms, the system must communicate with the user in domain-agnostic language. This includes avoiding use of jargon, and explaining any user choices in as intuitive a way as possible.
- **C5: Support Diverse Task and Data Types:** Supporting the autoML system over a large number of data types and problem types (Table 1), serves as a design challenge. A single visualization, such as a confusion matrix, could be used for multiple data types and problem types, while others require their own unique visualization. Interaction should be supported when possible, but it must be relevant to the user’s task and consistent with the capabilities of the autoML engine.

We address the design constraints in Snowcat by using a flexible, modular, browser-based frontend, a black box autoML system, and



a middleware server that communicates between them, based on the architecture illustrated in Figure 2(c).

## 6 THE SNOWCAT SYSTEM

As part of the DARPA “Data Driven Discovery of Models” (D3M) program [39], over the last year, the authors of this paper worked with groups of machine learning experts and domain scientists to develop a visual analytics system that interfaced with multiple autoML engines. The resulting VA system *Snowcat* serves as an instantiation of our proposed architecture and framework.

### 6.1 System Requirements

The goal of Snowcat is to allow a user to utilize the autoML systems to generate a complex machine learning model that will perform well on held-out test data. Many data types and problem types are supported (Table 1). The expected users of Snowcat are subject matter experts (SMEs) in the domain of the dataset, but no other stipulations about their proficiency at data science is made.

In Snowcat, a user first chooses a dataset, and then generates a relevant machine learning problem on that dataset. Then, Snowcat communicates the problem to several autoML systems, generating multiple complex models for the user to choose from. The user is then shown relevant information on each of the returned models, including estimations of their performance on held-out testing data and their predictions on validation data. The user is tasked with choosing the model that they believe the best for the task at hand. An express goal of the system is to demonstrate that the SME user has insight into the data that is not available to the autoML system, and thus the user should be given the ability to choose their preference from a set of  $n$  models rather than being forced to use a single model.

### 6.2 Snowcat Client Design

The front end of Snowcat has a wide set of responsibilities. It must allow the user to explore the **Data** and support the user in creating a well-defined **Problem Specification**, as per our proposed VA-autoML framework (Figure 2). A well-defined problem specification comprises at least the selection of problem type, target and predict features, and performance metrics. It also must provide **Visualizations** of model outputs to facilitate **Model Selection**.

To facilitate this large scope of varied requirements (C5), we implemented the system in *Vue.js*, a reactive JavaScript front end library in which data elements can be bound to DOM elements. As the internal state of the client application gets updated by asynchronous communication with the middleware, *Vue.js* dynamically updates the components displayed in the browser (C2). In addition, *Vue.js* encourages modular development via single file components, in which the HTML, CSS, and JavaScript of a single component of a web application are all defined in the same file. The various visualizations and workflow guidance in Snowcat can then be developed as isolated components (C1).

#### 6.2.1 Workflow Guidance

In order to guide SME users through the data analysis process (C3), Snowcat features workflow management capabilities that are visualized in the system. The client interface consists of two parts, as shown in Figure 1(a). The left-side *workflow-panel* reflects the realization of the proposed VA-autoML framework and shows the current level and status of workflow execution. The right-side *card-panel* consists of multiple *cards* where each card targets a particular objective (e.g., visualizing input dataset or output models so user can interact and explore, or helping the user define problem specification, etc.).

Examples of the workflow panel of Snowcat are shown in Figure 1(b) and (c) for **Problem Specification**, **Model Training**, and **Model Selection**. Each component in the blue-dashed box area of the operational framework (Figure 3) corresponds to some subset of the steps in the workflow-panel, which in turn corresponds to one or more cards. The workflow panel provides guidance and a functionality of backtracking and feedback loop (C3). Any previous step can be reached by clicking on the corresponding workflow step. The user is guided by simple and

clear instructions in the workflow panel (C4) that cue them to particular cards in the card panel via cross-highlighting.

#### 6.2.2 Modular Dynamic Content

The card panel gives the opportunity to develop the system in a modular way, as new *cards* can be developed or refined independent of previous cards (C1, C5). Data exploration is facilitated by various cards corresponding to different data types (e.g., the card in Figure 1(g) visualizes a tabular dataset, Figure 1(h) shows a card that visualizes a text dataset, and Figure 1(i) shows a card visualizing a graph matching dataset). Problem discovery (a step in **Problem Specification**) has its own card as well (Figure 1(e)). Once the user chooses a task and a dataset to explore, our system workflow shows (Figure 1(c)) the resulting problem specification (Figure 1(f)), data specification, and data visualization (e.g., Figure 1(g)) cards.

As autoML tools generate the resulting predictive models, we show details through some model output cards (e.g., a set of residual bar charts as in Figure 1(j), a set of confusion matrices such as in Figure 1(k), etc.). During any step in the workflow execution, users can add any card (e.g., data visualization or problem specification) through an option on the card panel (e.g., Figure 1(d) shows a possible list of cards), allowing the user to customize the content they see on Snowcat at any point in the workflow.

#### 6.2.3 Model Selection

Since support for model selection with autoML is limited, other features of autoML must be exploited in order to allow for a user’s domain expertise to improve on an autoML system’s automated selection. One such exploitation is to have the autoML system generate a multitude of diverse models and allow the user to choose a particular model that best matches their understanding of the data. The front end should then support comparison of models. However, comparing two models generated by different learning algorithms offers its own challenges. Thus, rather than comparing the internals of a model, Snowcat compares the models’ predictions on a held out validation set. By comparing multiple models’ predictions, the user can get a sense of the decision boundaries and sensitivities of each model. To facilitate model comparisons, Snowcat primarily uses juxtaposition of modular visualizations for each model returned by the connected autoML systems [14].

### 6.3 Snowcat Middleware

The middleware facilitates communication between the user and autoML, and it is responsible for initiating any data preprocessing. Because it must handle asynchronous communication with both the back-end and the client, it is of paramount importance that it can handle distributed processing (C2). For this reason, Snowcat uses *Node.js* as its middleware. *Node.js*’s event-based model makes it easy to handle different scales of communication with the client and the autoML component.

The middleware is also responsible for running any preprocessing tasks required by the client. Because different types of data may require different software packages, and because it is important that different client components can be developed in isolation, we allow for arbitrary middleware processing, including Python, MATLAB, C, or C++ scripts, by spawning *Node.js* child processes.

Snowcat uses Web Sockets to facilitate bi-directional communication between the middleware and the client. It is used by the middleware to receive problem specifications from the user, to serve data to the client, and to run preprocessing and steering queries on the data that correspond to the different cards shown in the client application. The middleware is likewise able to send updates on the status of models as they are being built by the autoML component, a key feature for dealing with scale (C2).

### 6.4 autoML System

The autoML system is treated as a black-box in Snowcat. It currently uses multiple experimental autoML engines developed as part of DARPA’s D3M program [39], which are planned to be open sourced by the conclusion of the program. Various teams of researchers including

Problem Types											
Data Types		Classification	Regression	Clustering	Link Prediction	Vertex Nomination	Community Detection	Graph Clustering	Graph Matching	Time Series Forecasting	Collaborative Filtering
	Tabular	✓	✓	✓	X	X	X	X	X	X	✓
	Graph	✓	✓	✓	✓	✓	✓	✓	✓	X	✓
	Time Series	✓	✓	✓	X	X	X	X	X	✓	✓
	Texts	✓	✓	✓	X	X	X	X	X	X	✓
	Image	✓	✓	✓	X	X	X	X	X	X	✓
	Video	✓	✓	✓	X	X	X	X	X	X	✓
	Audio	✓	✓	✓	X	X	X	X	X	X	✓
	Speech	✓	✓	✓	X	X	X	X	X	X	✓

Table 1. List of all problem types and data types supported by Snowcat . A check mark indicates if a problem type can be applied to a particular task type, while a cross mark is used to denote incompatible matching between data and task types.

the authors of this work contributed to the development of an open source API that communicates with a general autoML system [1]. This API is defined in Google’s implementation of the Remote Procedure Call protocol, gRPC [37]. gRPC allows for an API service to be defined in a meta-language that can be compiled down to many different languages, including JavaScript, Python, and C#. Data is structured in protocol buffers [2], which are language-neutral, platform-neutral, extensible mechanisms for serializing data.

## 7 USAGE SCENARIOS

In this section, we provide two usage scenarios with two real datasets. These two scenarios are chosen to demonstrate the utility of Snowcat with regard to its support of different data types, task types, and analysis procedures. The first scenario involves a time-series forecasting problem of milk price in a confirmatory data analysis context. The second scenario is a regression analysis for analyzing automobile fuel efficiency in an exploratory data analysis scenario.

### 7.1 Confirming the Rise in Wholesale Milk Price Forecasting

John has a business of making dairy products and is expecting a rise in the milk wholesale price in the coming season. He would like to confirm whether his speculation of a possible rise in the milk price is accurate, as it would affect the production cost in his business. John downloads the data of wholesale price of fluid milk over a timeline from the International Institute of Forecasters<sup>2</sup> [30]. Although he is familiar with the wholesale milk price data it is difficult for John to test his hypothesis due to his lack of expertise in data analysis and machine learning.

Using Snowcat John starts with exploring the downloaded fluid milk data in order to create a problem specification and to select parts of the data to be used in his analysis. For this, John starts with loading the **Data** in Snowcat, which is then shown through two interactive visualizations: a *tabular* form similar to Figure 1(g) where each histogram shows the distribution of data items over a set of ranges, and a *time-series line chart*, such as in Figure 4(a)), for each time-series data type in the input dataset to reflect its value changes over time.

John explores the data and wants to create a predictive model based on the milk price forecasting. He selects the milk price field (i.e., the *value* in the dataset) as a target feature and *time* as the predict feature to train the model. Snowcat then shows that there are two possible problem types for the selected target feature: regression and the time-series forecasting. As John is interested in confirming his hypothesis of rise in milk price, he selects the *time-series forecasting* as the problem type. He then picks the default performance metric *root mean squared error* as he is not familiar with other metrics. John submits the final



Fig. 4. The line chart shows predictions of the fluid milk price over time. (b) to (e) show the predictive values of the four generated models by the autoML system in Snowcat against the original values in (a). Through the use of this visualization, a user can compare the models alongside the return values of the performance metric to identify the most appropriate model.

**Problem Specification** (Figure 1(f)) to Snowcat. The autoML engine then uses this submitted problem specification and the selected data (e.g., the *value* and the *time* fields from the downloaded milk fluid dataset) for the **Model Training**.

The autoML engine returns four models alongside the value of the root mean squared error for each model. For **Model Selection**, Snowcat visualizes the generated models using time-series line chart to show the predictive values of each resulting model (Figure 4 (a) to (e)). These line charts are stacked under the time-series line chart (Figure 4(a)) of the selected target feature. The four models in Figure 4 (b) to (e) show the predictive values against the original values of the target feature in Figure 4(a). Snowcat provides the option to move up or down each of these line charts or zoom-in or out so that users can easily explore and compare between the different models. John explores these models and selects the model in Figure 4(e)) as the “best” model, as it has the best value of root mean squared error metric and predicted better than other models in the testing data. He asks the Snowcat to export this model

<sup>2</sup><http://forecasters.org/resources/time-series-data/>

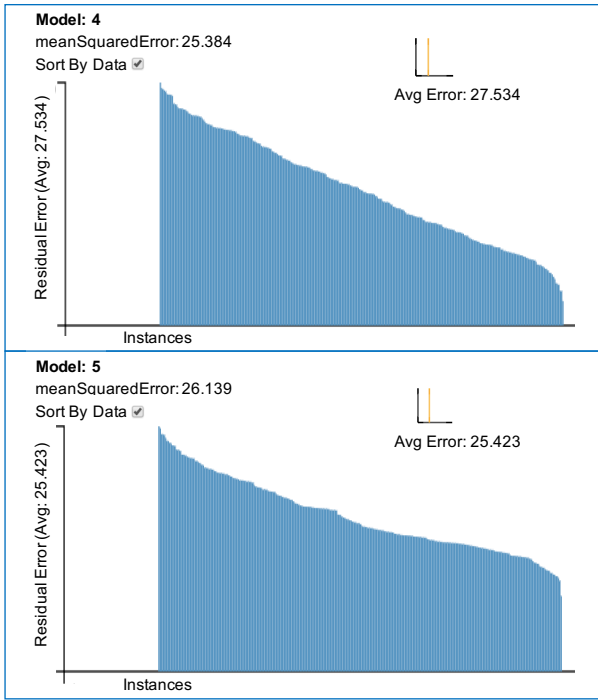


Fig. 5. Regression plots of the two best models returned by the autoML system in Snowcat. Note that model 4 has lower mean squared error than model 5, but it has more variance in its residuals, indicated by a higher slope.

and then uses this exported model and finds out that his speculation is true about the possible rise.

## 7.2 Exploring the Automobile Fuel Data to Understand Fuel Efficiency

Erica is an executive at a food delivery service company and she would like to do research what type of customized vehicle she should buy for her fleet of delivery workers. In particular, she wants insight into what measurable qualities of a car may affect its fuel efficiency, as fuel is one of her organization’s largest expenses. She hopes to create a regression model for miles per gallon (mpg) that can be used to assess new designs and prototypes of vehicles.

Erica has access to a dataset containing information about 398 cars (available from OpenML [47]), and she would like to build a set of predictive models using different sets of prediction features to determine which features may be most effective at predicting fuel efficiency. She begins by loading the **Data** and explores the relationship between attributes in the histogram view, seen in Figure 1(g). By hovering over the bars corresponding to mpg, she determines that the number of cylinders and the class may be good predictors. She moves on to create a well-defined **Problem Specification** to be used by the autoML systems in Snowcat. For this, she generates a regression problem that attempts to predict fuel efficiency using those two columns, and optimizes for mean squared error. She then kicks off the **Model Training** process by submitting the regression problem to Snowcat.

The autoML system returns 6 models. The two best models both have similar scores for mean squared error. For **Model Selection** for regression models, Snowcat gives a sense of how the different models apportion residuals by displaying a bar chart of residuals by instance, sorted by the magnitude of residual (see Figure 5). Erica views the residual plots for the two best models, and notes that, while the mean squared error of model 4 is lowest, model 5 apportions residuals more evenly among its instances. Based on her requirements, it is more important to have a model that gives consistently close predictions, rather than a model that performs well for some examples and poorly for others. Thus, by following the Snowcat workflow, Erica was able to

get a better sense of her data, to define a problem, to train a model, and to select the model that she believed would perform best for her usage scenario.

## 8 DISCUSSION

### 8.1 Low Floor Design

As the goal of an integrated VA and autoML system is to support the user in model discovery, it is important to guide the user, step by step, through their task to achieve their goal. The system must provide cues towards what remains to be done to finish their task. In the parlance of user experience, this system must have a low *floor*, or a gentle learning curve, addressing challenges **C1** and **C2**.

Because the variance of capabilities of potential users is so large, it is likely that there is more marginal utility designing for a low floor rather than a high *ceiling* of capabilities. As such, it may be wise for the system to be able to take the reins from the user if the user is not sure what to do next. In Snowcat, we call this the *Ship It!* functionality – at any step, the user can click through the workflow to request that the system automatically make the default choices, so that in the worst case of the user struggling with the data analysis process, Snowcat can at least produce a reasonably good model. In addition, since Snowcat allows the user to step backwards, the user can use the *Ship It!* functionality to move forward, see what Snowcat recommends, and then move backwards and try it out themselves.

### 8.2 Bridging the Gap Between autoML and User Intent

Since the autoML system adheres to a strict API, it may be tempting to build a system that simply exposes the exact API options, along with visualization support. However, there is often a gap between the user’s intent and the well-defined autoML API. The utility of a visual analytics system lies in bridging this gap, but in many cases, no perfect translation exists.

AutoML systems expect a problem to be well defined and precisely labeled, and they require a metric to define a cost function to optimize. In contrast, the user may have a non-computational notion of the data and the cost of a given model. When a user judges the predictions of a model, certain instances are more important to them than the others. Suppose a meteorologist is building a model to predict the occurrence of hurricanes. The false positive rate and false negative rate have drastically different costs, and that ratio must be encoded in the cost function optimized by an autoML system. A visual analytics system could attempt to infer the user’s cost function over model predictions, and help the user choose a model that performed best over that cost function.

In some cases, the value of instances varies not by false or true positive rate, but rather by the features of that instance. If a hospital administrator is building a triage model to determine which patients should be prioritized, they may feel that misclassifications for child patients are more egregious than for adults. A visual analytics system could allow the user to categorize data, and then train models on particular subsets that favor the data that is most important to the user.

## 9 CONCLUSION

In this work, we address the needs of VA systems that generate models using autoML as a black box. First, we present a VA-autoML framework, extended from classic frameworks for visualization and VA, that includes the use of an autoML system for data modeling. Second, we propose a client-server architecture for VA systems that follow this framework. This architecture relies on a middleware that controls communication between the client, the autoML backend, and the database via asynchronous two-way channels. To demonstrate the efficacy of our framework and architecture, we present a prototype VA system, Snowcat. We demonstrate the efficacy of Snowcat via two usage scenarios in which subject matter experts build a time series forecasting model and a regression analysis, respectively. By presenting a framework, an architecture, and a prototype system, this work lays the groundwork for visual analytics for automated ML, a field which has the potential to bridge the gap between cutting edge ML and the needs of domain experts.



## ACKNOWLEDGMENTS

This material is partially based on research sponsored by the Air Force Research Laboratory and DARPA under agreement number FA8750-17-2-0107

## REFERENCES

- [1] Grpc protocol definition for the ta2-ta3 communication api. <https://gitlab.com/datadrivendiscovery/ta3ta2-api>. Tag: v2017.12.20.
- [2] Protocol buffers - google's data interchange format. <https://github.com/google/protobuf>. Accessed: 2018-03-24.
- [3] N. Andrienko, T. Lammarsch, G. Andrienko, G. Fuchs, D. Keim, S. Miksch, and A. Rind. Viewing visual analytics as model building. *Computer Graphics Forum*, 0(0). doi: 10.1111/cgf.13324
- [4] L. Battle, M. Stonebraker, and R. Chang. Dynamic reduction of query result sets for interactive visualization. In *Big Data, 2013 IEEE International Conference on*, pp. 1–8. IEEE, 2013.
- [5] J. Bergstra, D. Yamins, and D. D. Cox. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in Science Conference*, pp. 13–20, 2013.
- [6] A. Bhardwaj, S. Bhattacharjee, A. Chavan, A. Deshpande, A. J. Elmore, S. Madden, and A. G. Parameswaran. Datahub: Collaborative data science & dataset version management at scale. *arXiv preprint arXiv:1409.0798*, 2014.
- [7] E. T. Brown, J. Liu, C. E. Brodley, and R. Chang. Dis-function: Learning distance functions interactively. In *Visual Analytics Science and Technology (VAST), 2012 IEEE Conference on*, pp. 83–92. IEEE, 2012.
- [8] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo. Vistrails: visualization meets data management. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pp. 745–747. ACM, 2006.
- [9] S. K. Card, J. D. Mackinlay, and B. Shneiderman. *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.
- [10] E. H.-h. Chi and J. T. Riedl. An operator interaction framework for visualization systems. In *Information Visualization, 1998. Proceedings. IEEE Symposium on*, pp. 63–70. IEEE, 1998.
- [11] J. Choo, C. Lee, C. K. Reddy, and H. Park. Utopian: User-driven topic modeling based on interactive nonnegative matrix factorization. *IEEE transactions on visualization and computer graphics*, 19(12):1992–2001, 2013.
- [12] K. A. Cook and J. J. Thomas. Illuminating the path: The research and development agenda for visual analytics. 2005.
- [13] M. Gleicher. Explainers: Expert explorations with crafted projections. *IEEE Transactions on Visualization & Computer Graphics*, (12):2042–2051, 2013.
- [14] M. Gleicher. Considerations for visualizing comparison. *IEEE transactions on visualization and computer graphics*, 24(1):413–423, 2018.
- [15] I. Guyon, K. Bennett, G. Cawley, H. J. Escalante, S. Escalera, T. K. Ho, N. Macia, B. Ray, M. Saeed, A. Statnikov, et al. Design of the 2015 chlearn automl challenge. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pp. 1–8. IEEE, 2015.
- [16] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.
- [17] F. Heimerl, S. Koch, H. Bosch, and T. Ertl. Visual classifier training for text document retrieval. *IEEE Transactions on Visualization & Computer Graphics*, (12):2839–2848, 2012.
- [18] J. M. Hellerstein, R. Avnur, A. Chou, C. Hidber, C. Olston, V. Raman, T. Roth, and P. J. Haas. Interactive data analysis: The control project. *Computer*, 32(8):51–59, 1999.
- [19] D. C. Hoaglin. John w. tukey and data analysis. *Statistical Science*, pp. 311–318, 2003.
- [20] G. Katz, E. C. R. Shin, and D. Song. Explorekit: Automatic feature generation and selection. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pp. 979–984. IEEE, 2016.
- [21] D. Keim, G. Andrienko, J.-D. Fekete, C. Görg, J. Kohlhammer, and G. Melançon. Information visualization. chap. Visual Analytics: Definition, Process, and Challenges, pp. 154–175. Springer-Verlag, Berlin, Heidelberg, 2008. doi: 10.1007/978-3-540-70956-5\_7
- [22] E. D. Keim, J. Kohlhammer, and G. Ellis. Mastering the information age: Solving problems with visual analytics, eurographics association, 2010.
- [23] B. Komer, J. Bergstra, and C. Eliasmith. Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In *ICML workshop on AutoML*, 2014.
- [24] 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. *Journal of Machine Learning Research*, 17:1–5, 2016.
- [25] Q. Le and B. Zoph. Using machine learning to explore neural network architecture. <https://research.googleblog.com/2017/05/using-machine-learning-to-explore.html>. Accessed: 2018-03-29.
- [26] F.-F. Li and J. Li. Cloud automl: Making ai accessible to every business. <https://www.blog.google/topics/google-cloud/cloud-automl-making-ai-accessible-every-business/>. Accessed: 2018-03-29.
- [27] M. Liu, J. Shi, K. Cao, J. Zhu, and S. Liu. Analyzing the training processes of deep generative models. *IEEE transactions on visualization and computer graphics*, 24(1):77–87, 2018.
- [28] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu. Towards better analysis of deep convolutional neural networks. *IEEE transactions on visualization and computer graphics*, 23(1):91–100, 2017.
- [29] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.
- [30] S. Makridakis and M. Hibon. The m3-competition: results, conclusions and implications. *International Journal of Forecasting*, 16(4):451 – 476, 2000. The M3- Competition. doi: 10.1016/S0169-2070(00)00057-1
- [31] P. Moritz and R. Nishihara. Plasma: A high-performance shared-memory object store. <https://arrow.apache.org/blog/2017/08/08/plasma-in-memory-object-store/>. Accessed: 2018-03-28.
- [32] T. Mühlbacher, L. Linhardt, T. Möller, and H. Piringer. Treepod: Sensitivity-aware selection of pareto-optimal decision trees. *IEEE transactions on visualization and computer graphics*, 24(1):174–183, 2018.
- [33] T. Mühlbacher and H. Piringer. A partition-based framework for building and validating regression models. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):1962–1971, 2013.
- [34] 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.
- [35] D. Ren, S. Amershi, B. Lee, J. Suh, and J. Williams. Squares: Supporting interactive performance analysis for multiclass classifiers. IEEE, August 2016.
- [36] D. Ren, S. Amershi, B. Lee, J. Suh, and J. D. Williams. Squares: Supporting interactive performance analysis for multiclass classifiers. *IEEE transactions on visualization and computer graphics*, 23(1):61–70, 2017.
- [37] L. Ryan. grpc motivation and design principles. <https://grpc.io/blog/principles>. Accessed: 2018-03-24.
- [38] D. Sacha, M. Sedlmair, L. Zhang, J. A. Lee, D. Weiskopf, S. C. North, and D. A. Keim. Human-Centered Machine Learning Through Interactive Visualization: Review and Open Challenges. In *Proceedings of the 24th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, Bruges, Belgium*, Apr. 2016.
- [39] W. Shen. Data-driven discovery of models (d3m). <https://www.darpa.mil/program/data-driven-discovery-of-models>. Accessed: 2018-03-24.
- [40] H. Strobel, S. Gehrmann, H. Pfister, and A. M. Rush. Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE transactions on visualization and computer graphics*, 24(1):667–676, 2018.
- [41] T. Swearingen, W. Drevo, B. Cyphers, A. Cuesta-Infante, A. Ross, and K. Veeramachaneni. Atm: A distributed, collaborative, scalable system for automated machine learning. In *Big Data (Big Data), 2017 IEEE International Conference on*, pp. 151–162. IEEE, 2017.
- [42] P. Tamagnini, J. Krause, A. Dasgupta, and E. Bertini. Interpreting black-box classifiers using instance-level visual explanations. In *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics*, p. 6. ACM, 2017.
- [43] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 847–855. ACM, 2013.
- [44] J. W. Tukey. *Exploratory data analysis*, vol. 2. Reading, Mass., 1977.
- [45] S. Van Den Elzen and J. J. van Wijk. Baobabview: Interactive construction and analysis of decision trees. In *Visual Analytics Science and Technology*

- (VAST), *2011 IEEE Conference on*, pp. 151–160. IEEE, 2011.
- [46] J. J. Van Wijk. The value of visualization. In *Visualization, 2005. VIS 05. IEEE*, pp. 79–86. IEEE, 2005.
  - [47] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013. doi: 10.1145/2641190.2641198
  - [48] G. Verbruggen and L. De Raedt. Towards automated relational data wrangling. In *Proceedings of AutoML 2017@ ECML-PKDD: Automatic selection, configuration and composition of machine learning algorithms*, pp. 18–26, 2017.
  - [49] E. Wall, S. Das, R. Chawla, B. Kalidindi, E. T. Brown, and A. Endert. Podium: Ranking data using mixed-initiative visual analytics. *IEEE transactions on visualization and computer graphics*, 24(1):288–297, 2018.
  - [50] X.-M. Wang, T.-Y. Zhang, Y.-X. Ma, J. Xia, and W. Chen. A survey of visual analytic pipelines. *Journal of Computer Science and Technology*, 31:787–804, 2016.
  - [51] K. Wongsuphasawat, D. Smilkov, J. Wexler, J. Wilson, D. Mané, D. Fritz, D. Krishnan, F. B. Viégas, and M. Wattenberg. Visualizing dataflow graphs of deep learning models in tensorflow. *IEEE transactions on visualization and computer graphics*, 24(1):1–12, 2018.