AFRL-RI-RS-TR-2019-077

EVE: A VIRTUAL DATA SCIENTIST (D3M/EVE)

CHARLES RIVER ANALYTICS, INC.

APRIL 2019

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

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 was cleared for public release by the 88th ABW, Wright-Patterson AFB Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2019-077 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
PETER A. JEDRYSIK
Work Unit Manager

/ S /
ROBERT MCHALE for
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

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
APRIL 2019	FINAL TECHNICAL REPORT	APR 2017 – JUN 2018

4. TITLE AND SUBTITLE

EVE: A VIRTUAL DATA SCIENTIST (D3M/EVE)

5a. CONTRACT NUMBER
FA8750-17-C-0120

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62702E

6. AUTHOR(S)

Mukesh Dalal, Avi Pfeffer, Brad Rosenberg, Josh Serrin, Jonathan Hsu, Alan Fern, Erich Merrill, Oregon State University, Jonathan Almeida, Nina Zumel, PhD, Win Vector

5d. PROJECT NUMBER
D3MP

5e. TASK NUMBER
S0

5f. WORK UNIT NUMBER
02

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Charles River Analytics Inc.
625 Mount Auburn Street
Cambridge, MA 02138

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RISB
525 Brooks Road
Rome NY 13441-4505

DARPA/I2O
675 N. Randolph St
Arlington, VA 22203-2114

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2019-077

12. DISTRIBUTION AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. PA# 88ABW-2019-1316
Date Cleared: 27 MAR 2019

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Subject matter experts (SMEs) attempting to solve real-world analytic problems face several challenges due to the lack of applied mathematics, statistics, and machine learning skills that data scientists possess. The goal of our TA2 effort under the DARPA D3M Program was to span this gap by using novel methods and automation to enable SMEs to act as their own data scientists. Our effort was designed to fuse data- and knowledge-driven approaches to produce a virtual data scientist we call Eve. To translate domain-expert intent into formal representations of learning problems, we built a problem representation system that deterministically converts TA3 inputs into computer-interpretable mathematical expressions. To efficiently search for and compose the sequences of machine learning steps that comprise learning plans, we built a Monte Carlo Discrepancy Search approach that explores the vast space of possible plans through efficient modification and testing of prior related and/or successful plans. Further, we enriched these plans by incorporating data preparation models, treating data preprocessing functions as operators to be planned in-line with learning operators.

15. SUBJECT TERMS
Machine Learning, Data Science Optimization Planning, Decision-Making Monte-Carlo Tree Search

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**PETER A. JEDRYSIK**
U	U	U	UU	34	19b. TELEPHONE NUMBER (*Include area code*) **N/A**

Table of Contents

List of Figures

List of Tables

1. Summary

A historical gap exists between subject matter experts (SMEs) who struggle to solve analytic problems about the real world (e.g., cyber threat detection, sales forecasting) and data scientists who have broader analytic capabilities in applied mathematics (e.g., machine learning and data transformations) and the ability to apply these sophisticated techniques to a wide variety of problem domains. The goal of the DARPA Data-Driven Discovery of Models (D3M) program is to span the gap by providing novel methods to help SMEs solve analytics problems without the need for a dedicated data scientist. The D3M consists of three technical areas: TA1, which develops machine learning primitives, TA2, which uses these primitives to develop learning plans for specific problems, and TA3, which interfaces with the user to formulate problems and present results.

Subject matter experts (SMEs) attempting to solve real-world analytic problems face several challenges due to the lack of applied mathematics, statistics, and machine learning skills that data scientists possess. The goal of our TA2 effort was to span this gap by using novel methods and automation to enable SMEs to act as their own data scientists. Current approaches to automate data science are often data-driven exercises, failing to incorporate the SME's domain knowledge. Instead, our D3M effort was designed to fuse data- and knowledge-driven approaches to produce a virtual data scientist we call Eve. To translate domain-expert intent into formal representations of learning problems, we built a problem representation system that ingests TA3 inputs as computer-interpretable mathematical expressions defining problems to be solved. To efficiently search for and compose the sequences of machine learning steps that comprise learning plans, we built a Monte Carlo Discrepancy Search approach that explores the vast space of possible plans through efficient modification and testing of prior related and/or successful plans. Further, we enriched these plans by incorporating data preparation models, treating data preprocessing functions (e.g., handling missing data and dimensionality reduction) as operators to be planned in-line with learning operators. Combined, these capabilities will allow Eve to provide SMEs with a virtual data scientist to help address complex analytical learning problems without assistance from professional data scientists.

During Year 1 of the Eve project, we:

- Designed and implemented the Eve Search Space, including representations for data modeling primitives, composition operators, planning goals, and complex modeling pipelines.
- Documented and refined the technical requirements to integrate with new TA1 primitives and interface with TA3 components.
- Designed, implemented, and refined the planning and optimization component that transforms data into models that meet SME needs, including selection and configuration of primitives.
- Integrated the Eve automated model composition software by developing and refining APIs for integration, integrating Eve TA2 elements in preparation for integration events, and participating in semi-annual D3M integration events with TA1 and TA3 performers.
- Worked with SMEs and other D3M performers to exercise and evaluate the D3M toolkit's ability to build complex data modeling pipelines for user-specified data and outcomes of interest.

The benefits of a complete Eve project would include the ability to fuse data- and knowledge-driven approaches to imitate data scientists, a just-in-time mathematical ontology to construct fully formed learning concepts, the ability to plan using a Monte-Carlo Discrepancy Search to develop candidate solutions, and treating data preparation as a top-level planning element.

2. Introduction

2.1 Problem Description

A historical gap exists between subject matter experts (SMEs) who struggle to solve analytic problems about the real world (e.g., cyber threat detection, sales forecasting) and data scientists who have broader analytic capabilities in applied mathematics (e.g., machine learning and data transformations) and the ability to apply these sophisticated techniques to a wide variety of problem domains. The goal of the DARPA D3M program is to span the gap by providing novel methods to help SMEs solve analytics problems without the need for a dedicated data scientist.

Data scientists must work with SMEs to understand their domain knowledge, explore and process the data used to learn models, and manage the statistical and machine learning (ML) methods used to learn them. Current approaches to automating machine learning, such as the Automatic Statistician (Lloyd, Duvenaud, Grosse, Tenenbaum, & Ghahramani, 2014) and Auto-WEKA (Thornton, Hutter, Hoos, & Leyton-Brown, 2013), focus on learning statistical models from data and ignore the domain knowledge provided by the SME. Because of the complexity of most real-world data-science problems, these methods tend to miss key relationships, find spurious ones, and produce learned models that are not understandable by humans. Moreover, the pre-processing that must occur before machine learning algorithms can be run relies heavily on an SME's domain knowledge to ensure the semantics of the SMEs problem are not altered. Our fundamental insight is that by using a well-designed knowledge representation framework, we can create a unified interface to the domain, data, and statistical models to take advantage of the SME's domain knowledge and the data to create effective machine learning models that support the SME's analyses. To achieve the TA2 objectives of the D3M program, we proposed to produce a knowledge- and data-driven virtual scientist called Eve.

To develop Eve, we had to address the following two major challenges:

- Eve must efficiently search for and compose abstract plans of data preparation and machine learning operators (high-level manipulations that must be made) to solve the specified learning problem.
- Eve must select effective primitives to implement these operators, composing executable plans that address both learning objectives and constraints, and the prioritized metrics identified by the SME.

Regarding the first challenge—that Eve must efficiently search for and compose abstract learning plans to solve the specified learning problem—we face two key sub-challenges. The first, a planning sub-challenge, is how to select from a potentially enormous number of data preparation, transformation, and learning operator sequences that are consistent with solving a specified learning problem. Our approach was to use techniques from the planning and reinforcement learning literature, such as the highly successful Monte-Carlo Tree Search (MCTS) (Browne et al., 2012), which was used in the AlphaGo system that recently defeated a grandmaster at Go. Eve also develops a novel form of MCTS, called Monte-Carlo Discrepancy Search, which improves already evaluated plans by introducing discrepancies/modifications into those plans. For example, if a plan uses polynomial interpolation as a data transformation step, a discrepancy may be to use linear interpolation instead, but otherwise follow the current plan. To seed this learning process, Eve draws from a library of historical plans from previous Eve applications, previous plans and plan templates constructed by data scientists in other learning applications, and offline learned plans from previous tools constructed by our team.

The second sub-challenge is how to automate data preparation that often consumes most of the resources of typical data science projects (Zumel, Mount, & Porzak, 2014). Data scientists understand that the choices made in preparing the data (e.g., how to represent categorical data, how to handle missing data) can dramatically affect the performance of machine learning components. Our approach treated all data preparation functions as operators to be planned in conjunction with the learning operators. This enabled us to create sequences of data preparation operators integrated with learning operators in a single framework, which is possible because of our unified representation of the data and statistical models.

Eve's second challenge—selecting the optimal primitives to transform initially abstract plans into fully-specified learning programs—amounts to compiling a declarative representation of an abstract plan into a procedural representation of how to do it (e.g., by choosing particular classifiers or optimizing hyper-parameters). The search for specific primitives and hyperparameter values is integrated into our planning process, using a Monte Carlo Discrepancy Search similar to the one used to address the first challenge.

2.2 Technical Approach

Eve's goal is to enable subject matter experts (SMEs) to solve analytic problems by enabling them to be their own data scientists. For example, a cyber analyst may be tasked with predicting trends in malware. This may require featurizing a database of malware, clustering the malware into families, and learning a time series model of family-level features. A space analyst may face different challenges, such as how to classify space objects (SOs) to enhance space situational awareness. In this case, many algorithms may be used to resolve and correlate conflicting data against known SOs to identify potential threats.

To automatically solve different data science problems, our technical approach was grounded in cognitive systems engineering (CSE), advanced knowledge representation techniques, automated planning, and machine learning. In our original plan, which included a TA3 service, we would employ established principles for joint human-automation teaming to ensure that Eve's human-model interactions address and intuitively guide key user activities, including problem characterization and refinement, process selection and editing, and model review and curation. Second, we use advanced knowledge representation to relate and translate between the SME's domain model, the data, model and the learning model. Third, we frame the data science process as a planning and optimization problem, in which we use machine learning and search algorithms to build abstract plans of optimal operator sequences and concrete executable plans of primitives. When our scope was revised to only include TA2, the first two parts of the plan were deemphasized.

In our original design, Eve is organized into four process abstraction layers shown in Figure 1. The organization manages asynchronous complexities between layers, allowing for both top-down and bottom-up interaction, and enables plug-and-play interoperability with other performers' components. Given a domain problem, each successive layer produces an increasingly refined representation of the problem until such time as a machine learning plan may be executed or additional inputs are required of the SME. Consider an ecologist who wants to predict bird population (BP) for the upcoming August in Oregon to determine migration patterns. All four of Eve's layers work together and communicate closely to achieve this objective. The SME first interacts with Eve through the Human- Model Interaction (**HMI) Layer**. Since this layer is specific to TA3, we do not describe it here.

The Formal Modeling (FM) Layer receives the SME's representation from the HMI layer and deterministically translates the information supplied into formal mathematical expressions that can be assembled into integrated learning problem definitions and assessed for consistency and completeness. Benefits of the layer include its abilities to automatically join data sets (e.g., weather and population data sets with different spatial and temporal resolutions), capture relationships and constraints (e.g., temperatures are highest and rainfall is lowest in Oregon in August, which is most appealing to birds), and provide heuristics to guide planning for a solution (e.g., the relationship between temperature and GP is linear, which can be used as an initial starting point for search). The FM layer will guide the HMI layer's dialog with the SME. If the problem is not yet sufficiently specified, the HMI layer will be guided to elicit additional information from the SME before passing the problem definition down another a layer for planning. It will also communicate learning plans discovered by lower layers to the user. In doing so, it will translate between the mathematical constructs used in those plans and the SME's domain terms. While largely sitting in TA3, the FM layer is also required for TA2 to represent the problems and data to be worked on. Therefore, our effort included a limited development of the FM layer.

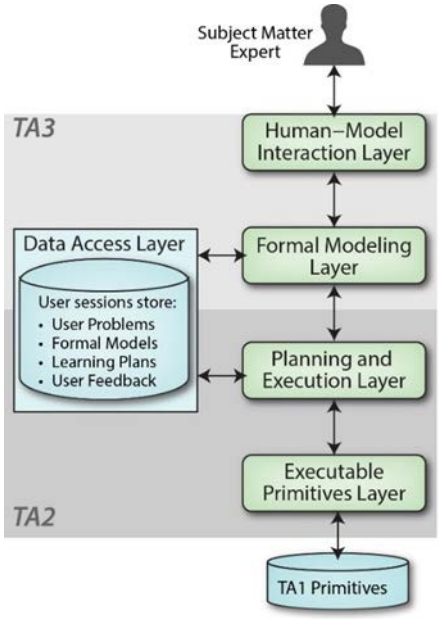

Figure 1: Eve system architecture

The Planning and Execution (P&E) Layer is responsible for searching for and composing abstract learning plans that are consistent with the FM layer's problem definition. These abstract plans provide the sequences of high-level data preparation, transformation, and planning operators that must be executed to successfully build a model addressing the problem. Operators provide a class of functions that can be addressed using a variety of primitives. The P&E Layer holds a large set of operators, including classes for all primitives developed by TA1 performers (e.g., imputation of missing data, conversions, transformations, regressions, time-series models, etc.). It also holds a library of learning plan templates that can be instantiated for problems of the appropriate type (e.g., prediction) to produce initial plans. From there, the planner searches through different sequences of operators to improve potential plans. For example, to impute missing rainfall data, the planner may identify the planning sub-sequence *Remove Outliers* followed by *Interpolate Remaining Points.* Further, to predict BP in August 2017, it may construct the sub-sequence *Regress GP Average on GP, rainfall, and temperature from previous months* followed by *Learn Time Series Model of Yearly BP with monthly GP and yearly average temperature as exogenous inputs.* This plan is shown in Figure 2; note that the green elements within this plan represent the abstract plan, while the orange elements represent concrete executable primitives for specific steps.

The P&E layer produces multiple plans that are passed down to the next layer for selection and configuration of specific primitives for each operator. For example, an alternative plan could be to learn a joint dynamic Bayesian network model over BP and GP rather than separate models.

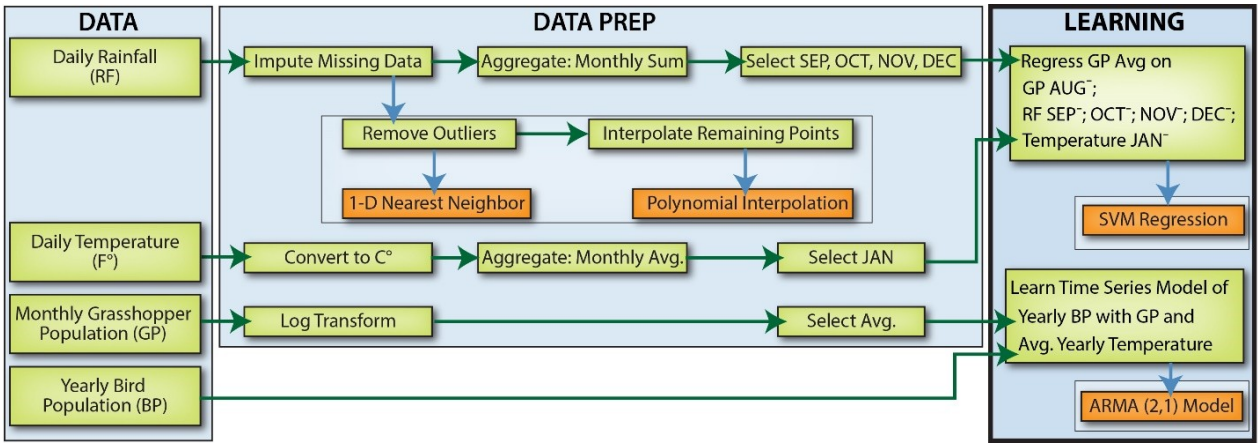

Figure 2: Example learning plan for bird population prediction

The **Executable Primitives (EP) Layer** chooses and refines machine learning primitives for each data preparation and planning operator and sends them up to the P&E layer to build a concrete (or executable) plan. In addition to selecting primitives, this layer explores and optimizes hyper-parameter variations (e.g., the number of autoregressive and moving average terms in the Autoregressive Moving Average (ARMA) model). The EP layer selects these primitives based on their performance with respect to targeted user and automated metrics for evaluating predicted model success. In the plan shown, the *Remove Outlier* operator may be assigned a *One-Dimensional Nearest Neighbor* algorithm as its primitive and the *Interpolate Remaining Points* operator may be assigned *Polynomial Interpolation*. For the *Regression* operator, Eve can select from a variety of primitives, such as linear, support vector machine (SVM) regression, or neural networks based on the domain of discourse, SME-specified metrics, and learned quality. In this case, *SVM Regression* is selected. Lastly, the primitive selected for *Learn Time Series Model* is an *ARMA Model*, with 2 and 1 as its chosen hyper-parameters.

Each of these plans will be effectively rated, before being posed to the SME for prioritization and selection. The HMI layer can display this information about the expected performance of these plans. By observing the tradeoff between these plans, the SME can choose to execute plans that meet the requirements most important to them.

During Year 1 of our effort, we began to realize our design, focusing on the Formal Modeling Layer, Planning and Execution Layer, and Executable Primitives Layer. For the Formal Modeling Layer, we created a mechanism to take in problem descriptions from TA3 and to return executable pipelines to TA3. For the Planning and Execution Layer, we developed a Monte Carlo discrepancy search algorithm to search for appropriate pipelines. For the Executable Primitives Layer, we ingested TA1 primitives, selected appropriate primitives to solve problems, along with hyperparameter values, and used those primitives in the pipelines we returned to TA3.

3. Methods, Assumptions, and Procedures

3.1 Meetings and Presentations

Table 1 outlines the meetings and presentations held during the Eve project.

Table 1: Meetings and Presentations for the Eve Project

Meeting Name	Dates	Description
Internal meeting at Charles River Analytics, Cambridge, MA	3,9,19 May 2017	Discussed Eve status updates, technical matters, and planned next steps with the Charles River internal team
Bi-weekly Charles River/OSU Eve team meetings, telecon	May – September 2017	Discussed Eve status updates, technical matters, and planned next steps with the Charles River internal team and OSU
D3M-wide collaboration meetings	May – September 2017	Developed and reviewed D3M-wide requirements and designs
Requirements and design meetings with other performers and Government teams, telecon	3, 10, 16, 17 May 2017	Developed and reviewed D3M-wide requirements and designs
Collaboration with specific performers, telecon	15 June 2017	Collaborated with Remco Chang and Dillon from Tufts
On-demand Charles River Analytics and Win Vector Meeting, telecon	16 June 2017	Discussed Eve status updates, technical matters, and planned next steps with the Charles River internal team and Win Vector
Collaboration with specific performers, telecon	23 June 2017	Collaborated with James Honaker from Harvard and Vito D'Orazio from UT, Dallas
DARPA D3M integration event in Arlington, VA	11 September 2017 – 11 October 2017	Held an integration discussion, and had a dry run for evaluation
D3M Evaluation Event, Arlington, VA	29 January 2018 – 9 February 2018	Participated in evaluations of D3M systems
On-site visit at Charles River Analytics, Cambridge, MA	18 April 2018	Held a site visit with the Sponsor, Charles River, and OSU.

3.2 System Architecture

We designed an initial, high-level Eve architecture that allowed all components to interact with each other using a representational state transfer (RESTful) application programming interface (API). The components in this initial architecture are shown in Figure 3.

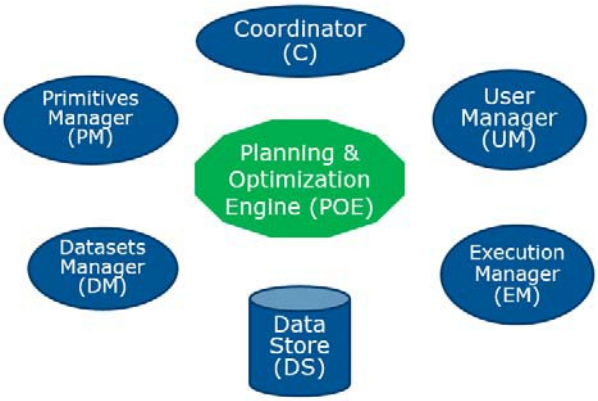

Figure 3: Initial Design of Eve Components

Within Eve, the Coordinator (C) component gets a command, builds a solution, and then delivers it. The Primitives Manager (PM) gets a primitive discovery request, and returns a ranked list of matches. The PM can also receive feedback on primitive quality to improve future results. The User Manager (UM) gets the challenge problem/refinement, and returns the optimization problem. The UM can also receive feedback on solution quality to improve future results. The Datasets Manager (DM) gets data. The Data Store (DS) provides storage for all the Eve components. The Planning and Optimization Engine (POE) produces ranked pipelines for a given challenge problem and produces a full model of the given pipeline. Each component has an interface used within Eve (API or other), a representational state transfer (RESTful) API, and a JavaScript object notation (JSON) payload. Oval components also have an external interface. Charles River worked on developing the blue components, and OSU the green component. We integrated Charles River and OSU components, and extended support for TA1 and TA3 APIs. We also integrated Eve components into two TA3 systems.

We then updated our architecture as shown in Figure 4.

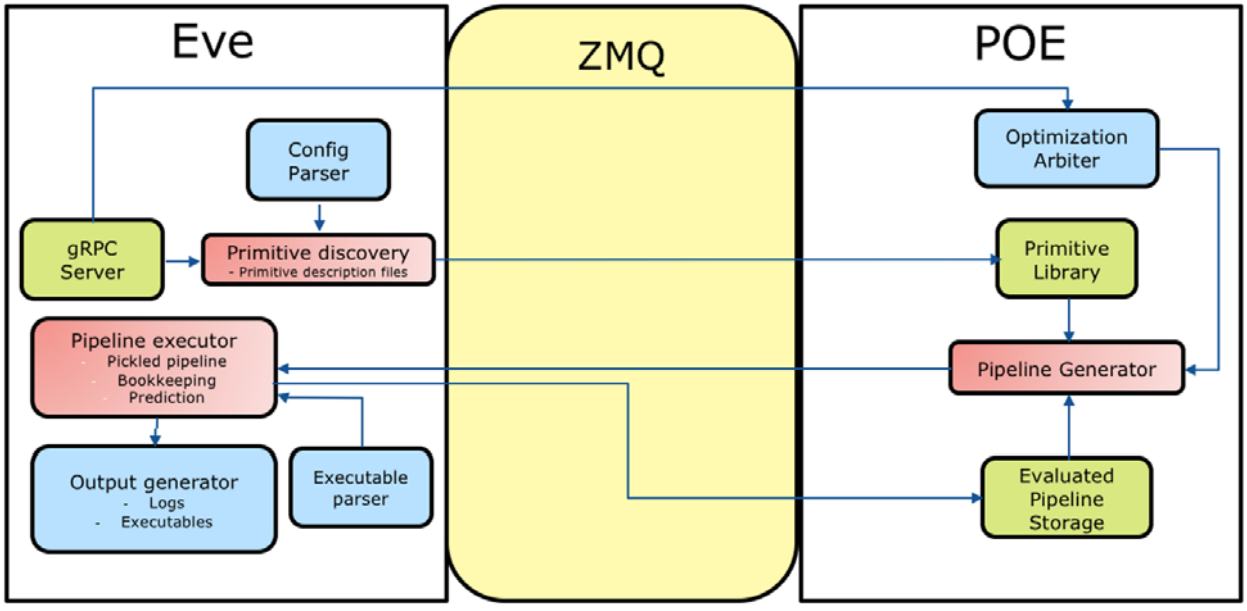

Figure 4: Year 1 Eve Architecture

This architecture consists of two main parts: Eve and the Pipeline Optimization Engine (POE). Eve interfaces with the rest of the D3M system, processing requests from the user via TA3, discovering TA1 primitives, and returning them back to the user via TA3. The POE is the component that executes the search for pipelines and hyperparameters. Eve and POE communicate via a Zero Messaging Queue (ZMQ).

Eve receives inputs from TA3 via its gRPC server. When it receives a problem, Eve discovers TA1 primitives using their Primitive Description Files. These discovered primitives are added to the Primitive Library for use by POE in its search for pipelines. Eve also informs POE's Optimization Arbiter of the goals of the optimization, so it can decide whether to accept a particular pipeline.

POE then goes to work. The Pipeline Generator generates pipelines using its discrepancy-based search algorithm, whereby it tries to generate algorithms that are similar to but improve on existing pipelines. These generated pipelines are sent back to Eve for execution and evaluation. POE uses the results of this evaluation to choose between pipelines. All the pipelines that have been evaluated are stored in the Evaluated Pipeline Storage. The Optimization Arbiter decides when to stop searching and which pipeline to return.

Eve then provides the best pipeline to TA3. It also provides a service to TA3 to execute this pipeline. Execution produces a pickled pipeline that can be saved for future use, various bookkeeping information, and a prediction for the user's problem. The output generator also creates logs, as well as an executable of the pipeline that TA3 can use later.

We developed a sequence diagram depicting how interactions were implemented. This diagram, shown in Figure 5, shows how Eve is used to define challenge problems, get and provide datasets to the Pipeline Optimization Engine (POE), and receive a list of pipelines from the POE.

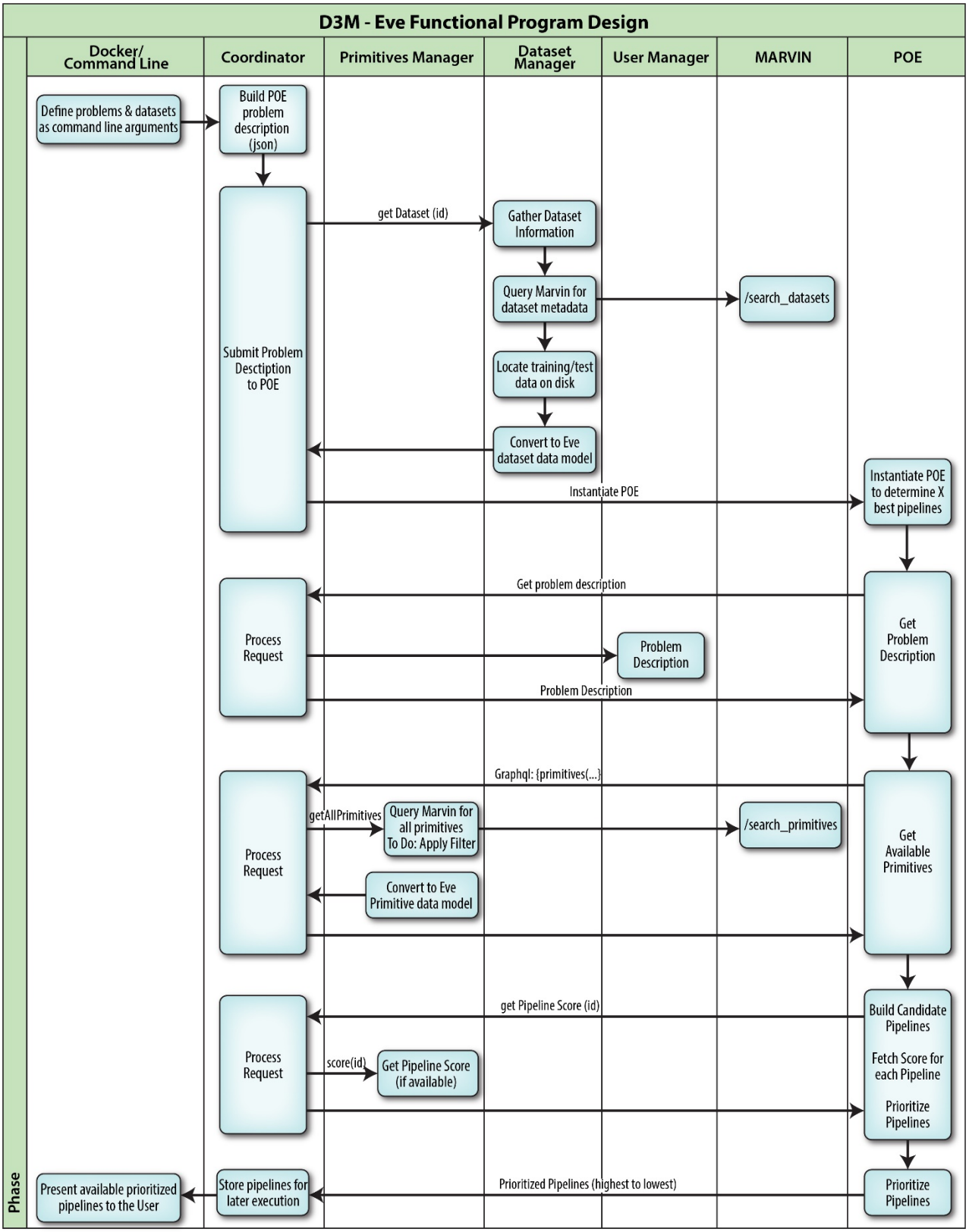

Figure 5: Sequence Diagram

Figure 6 shows Eve's dataflow when interacting with the user.

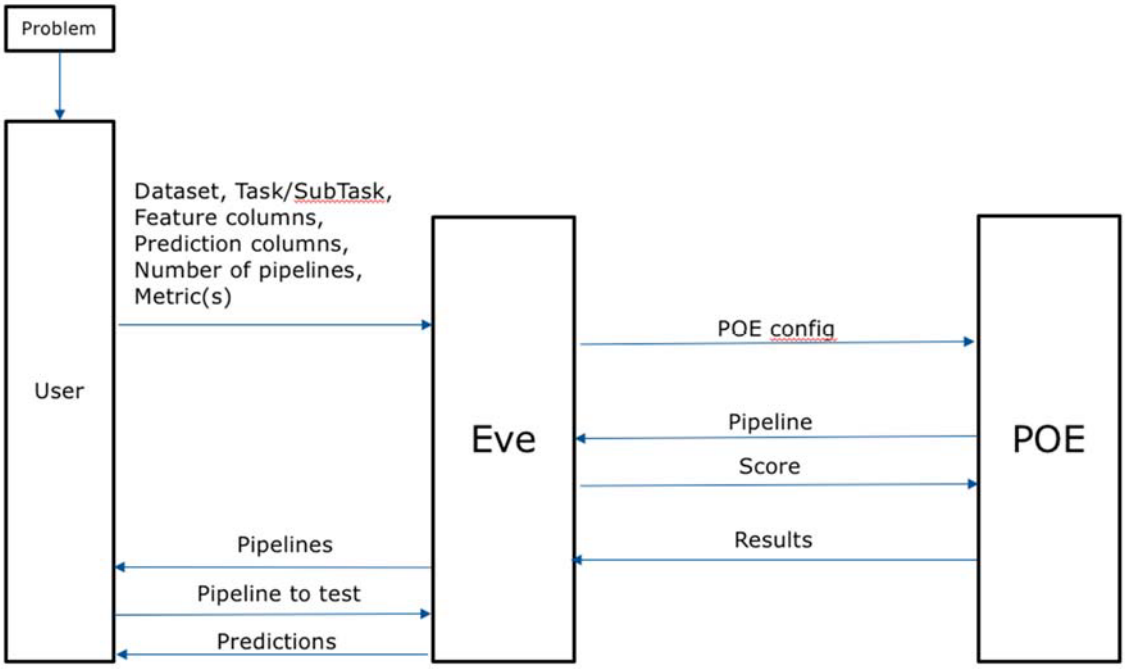

Figure 6: Eve Dataflow when Working with a User

The user works with TA3 to formulate a problem. TA3 sends details of the dataset, tasks and subtasks to execute, a description of feature columns and prediction columns in the data, a requested number of pipelines to generate, and one or more metrics to evaluate those pipelines. Eve works with the POE to generate the pipelines to solve the user's problem. Eve first configures the POE. The POE then repeatedly searches for and generates pipelines and sends them to Eve. Eve executes the pipelines and gives them a score according to the metrics, returning the score to POE, which uses it to guide its search. When POE has completed its search, it sends the results back to Eve, which returns the pipelines to the user via TA3. This completes the pipeline generation phase. In operation, once the user has selected a pipeline, he or she will want to run it on the data to generate results. Thus, the user sends a pipeline to test back to Eve. Eve runs it and sends predictions back to the user.

Figure 7 shows the Eve dataflow when interacting with the National Institute of Standards and Technology (NIST) test harness for evaluation.

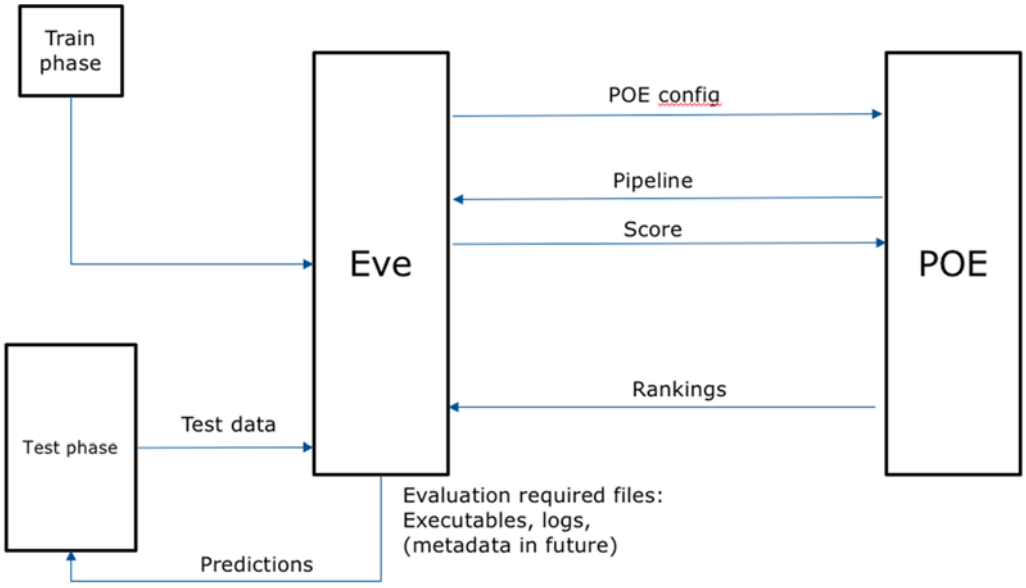

Figure 7: Eve Dataflow for the NIST Evaluation

Most of the workflow is similar, but the interactions outside of Eve are different. At the beginning, NIST initiates a training phase. Eve configures the POE, receives a sequence of pipelines from POE, scores them, and finally receives a ranking of the top pipelines from POE when the search is complete. Eve then produces the required artifacts for evaluation. The principal artifact is an executable file implementing the top pipeline. Eve also produces logs of its work and is also designed to produce metadata in the future. NIST then instructs Eve that a test phase has begun, and supplies the test data. The generated executable is used to process the test data and generate predictions. These predictions are then evaluated by NIST.

3.3 Formal Modeling Layer

During Year 1 of the Eve project, we designed and implemented the Eve Formal Modeling Layer, including representations for data modeling primitives, composition operators, planning goals, and complex modeling pipelines. We were active participants in TA1/2 and TA2/3 working groups that defined the APIs between the different TAs. Based on these APIs, we defined internal data structures to represent primitives and hyperparameter specifications coming from TA1, planning goals and data sources coming from TA3, executable pipelines passed to TA3, as well as the composition operators used to construct these pipelines.

As integration of the different capabilities is crucial to the success of the D3M program, we strived throughout our time on the program to work with other performers' components rather than develop our own substitutes. For example, we relied on primitives provided by the program. We developed an ability to automatically search for and discover TA1 primitives, read their metadata, and integrate the code in our search and planning process. We could ingest the primitives and run them dynamically. However, for the most part we had an incomplete set of primitives to solve problems.

Unfortunately, our automatic discovery of the primitives revealed a number of deficiencies and issues with the metadata. For example, certain combinations of hyperparameters caused certain primitives to crash, despite being legal according to the metadata. We also had incomplete metadata, where the ranges of hyperparameters were not specified, for example, the number of neighbors in a k-nearest neighbor search was not properly defined as an integer. The input/output type specification was also generally inadequate and inconsistent across TA1s. For example, we did not know the format for the array for the NumPy array. We could not fully rely on well-annotated metadata, and planned to possibly experiment with and curate TA1 primitives ourselves before evaluations. The unreliability of metadata about TA1 primitives presented an issue for the program, as it was difficult to automatically search for TA1 primitives. In addition, when we integrated with TA1 primitives, automatically searching for the primitives, we discovered and reported bugs with the primitives that prevented them from executing correctly with hyperparameter settings they claimed to handle. Working with other performers, we were working towards a resolution of these issues during our time on the effort.

At some point in our development it became clear that relying on the discovery and use of primitives from TA1 teams would be extremely risky. We then switched over to using primitives that were from the sklearn package due to their consistency and intercompatability, with a few manually-created internal primitives used to process some more obscure data types. Table 2 shows the final set of primitives we used in our later system evaluations, including each primitive's supported problem types, number of hyperparameters, and the types used to represent those hyperparameter values. Using these primitives was still quite challenging and required a large engineering effort, due to the changing API and many incompatibilities discovered during our testing, even for the sklearn primitives.

In hindsight, we believe we would have been better off to focus on using only our own internal set of primitives from the start of the project, which is the approach used by some other teams. We believe that our performance would have been on par with other teams in that case.

Table 2: List of primitives considered when constructing a full pipeline

Internal Name	Supported problem types	# Hyperparams	Hyperparameter types
sklearn.neighbors.NearestCentroid	Classification	0	
sklearn.linear_model.SGDClassifier	Classification	10	String, Boolean, Real
sklearn.svm.LinearSVC	Classification	4	String, Real
sklearn.ensemble.ExtraTreesClassifier	Classification	5	String, Boolean, Integer, Real
sklearn.ensemble.RandomForestRegressor	Regression	6	String, Boolean, Integer, Real
sklearn.decomposition.PCA	Classification, Regression	1	Real
sklearn.neighbors.KNeighborsClassifier	Classification	3	String, Integer
sklearn.ensemble.GradientBoostingRegressor	Regression	9	String, Integer, Real
sklearn.decomposition.KernelPCA	Classification, Regression	5	String, Integer, Real
sklearn.linear_model.PassiveAggressiveClassifier	Classification	3	String, Boolean, Real
sklearn.linear_model.LinearRegression	Regression	2	Boolean
sklearn.neighbors.KNeighborsRegressor	Regression	3	String, Integer
sklearn.naive_bayes.MultinomialNB	Classification	2	Boolean, Real
sklearn.naive_bayes.BernoulliNB	Classification	2	Boolean, Real
sklearn.linear_model.ARDRegression	Regression	5	Real
sklearn.cluster.FeatureAgglomeration	Classification, Regression	2	String, Integer
sklearn.ensemble.RandomForestClassifier	Classification	6	String, Boolean, Integer, Real
sklearn.naive_bayes.GaussianNB	Classification	0	
sklearn.kernel_approximation.RBFSampler	Classification, Regression	2	Integer, Real
osu_primitives.CategoryEncoder	Classification, Regression	0	
sklearn.ensemble.RandomTreesEmbedding	Classification, Regression	4	Integer
sklearn.decomposition.FastICA	Classification, Regression	4	String, Boolean, Integer
osu_primitives.TextEncoder	Classification, Regression	0	
sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis	Classification	1	Real
sklearn.tree.DecisionTreeClassifier	Classification	4	String, Integer, Real
sklearn.ensemble.ExtraTreesRegressor	Regression	5	String, Boolean, Integer, Real
sklearn.linear_model.SGDRegressor	Regression	10	String, Boolean, Real

3.4 Search and Planning to Discover Pipelines

3.4.1 Requirements Analysis for Discovery of Optimal Models

During Year 1 of the Eve project, we documented and refined the technical requirements to integrate with new TA1 primitives and interface with TA3 components.

Win Vector provided an advisory data science workflow to use as context when designing the Eve workflow and modeling primitives. They suggested key characteristics of primitives, tasks, and data for the purposes of workflow sequencing, primitive search, and discovery. The TA1 primitives are still not available.

We extended the prototype to execute the following workflow (POE refers to the planning and execution engine, while Eve below refers to the rest of the system):

1. Eve gets problem
2. Eve constructs a POE specific JSON file, with relative pathing
3. Eve starts up server
4. Eve passes JSON file to POE executable
5. POE requests pipeline run from Eve
6. Eve invokes dynamic script to run pipeline (which does data cleaning, but only run for the very first pipeline. A flag would be set internal to Eve and we would skip data cleaning primitives after that as rewashing so many times doesn't make sense)
7. Eve returns score to POE
8. Repeat until time limit from config is met by POE
9. Eve parses the answers CSV file for the top 20 UUID pipelines, and constructs logs and executables

Eve's dataflow when interacting with the user is described in Figure 6 above, while Eve's dataflow for the NIST evaluation is shown in Figure 7.

3.4.2 Planning and Optimization Engine

During Year 1 of the Eve project, we designed, implemented, and refined the planning component that transforms data into models that meet SME needs. We also designed, implemented, and refined the optimization component that selects primitives to guide plan execution. We combined these two components into a Planning and Optimization Engine (POE).

We refined our models based on the data on which TA2 operates. In particular, we studied methods to process multiple different data types, such as images and text, in the same problem, and how to combine their representations to apply a search algorithm. We also studied how to expand our pipeline framework to pipeline graphs rather than linear pipelines. We were able to process tabular and image data types; and handled binary and multiclass classification problems and univariate regression problems.

3.4.2.1 Prediction Pipeline Structure

Participating in the DARPA project gave us the opportunity to build a system designed to solve large, expensive-to-process, heterogeneous datasets representing challenging machine learning problems. Due to the size and complexity of the target problems, our chosen approach separated its machine learning pipeline search into two stages:

1. **Encoding Pipeline.** The encoding search considers different methods of transforming the

dataset's raw data into a numeric vector. This can involve simple transformations like one-hot encoding of categorical features, or more complex actions such as processing multiple types of raw data in parallel and considering different transformations and methods for combining the result in a semantically-appropriate way.

2. **Prediction Pipeline.** The search considers different prediction methods and their parameterizations, each taking as input the vector resulting from an encoding pipeline.

Figure 8 illustrates this two-stage prediction architecture which our system must search over. The first stage includes encoders for each type of input data (e.g. text, images, audio, vectors) and a combination method for the encodings (e.g. concatenation). The second includes one or more predictors and combining methods for their predictions. One of the key challenges in this search is that it combines search over a combination of discrete and continuous/numeric choices. For example, the choice of which encoders and prediction primitives to use are discrete choices, while the selection of parameters used by those primitives may be either discrete or continuous.

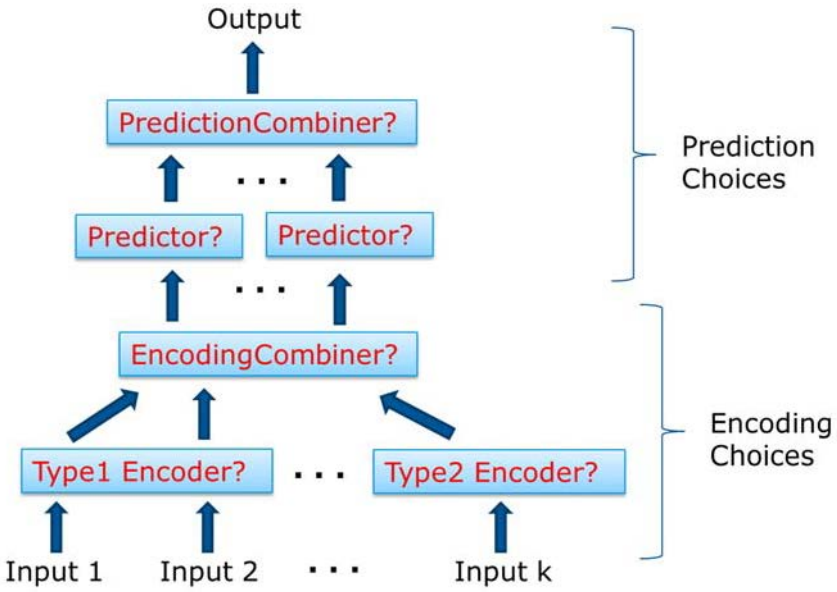

Figure 8: Depiction of the two-stage prediction architecture used in our system.

3.4.2.2 Search Approach

Our analysis of existing autoML systems revealed that the most successful approaches can be viewed as keeping track of individual pipelines and racing them against each other and small variations of the tracked pipelines. While most autoML systems were not described in this way, we decided to make that approach explicit in our system. In particular, our search approach can be viewed as a type of discrepancy search, where we maintain a set of pipelines and iteratively apply discrepancy operators to a selected pipeline in hopes of improving it.

To describe our approach, it is useful to distinguish between the *structure* and *parameters* of a concrete pipeline.

- **Pipeline Structure** – a directed acyclic graph where the vertices are primitives and edges indicate input-output relationships among primitives. Figure 8 shows an example of a Pipeline Structure, noting that the nodes would normally be labeled by specific primitives rather than generic primitives.
- **Pipeline Parameters** – given a pipeline structure each node primitive has an associated set of hyperparameters associated with it. The pipeline parameters are an assignment of values to all of those hyperparameters.
- **Full Configuration** – the combination of a pipeline structure and associated pipeline parameters. A full configuration can be executed to learn a model for a given training set and then be applied to test data.

Our goal is to identify a high-performing full configuration by searching through the enormous space of configurations. Our high-level strategy decomposes the search into a search over structures and then over parameters for those structures. This decomposition is motivated by the empirical observation that often selecting the correct structure is a dominating factor in performance and that the right structure with default parameters will typically outperform a lesser structure with highly optimized parameters. This suggests that identifying good structures should be a priority, with parameter optimization playing a secondary, though important, role.

Multi-Armed Bandits. More formally, we draw on the framework of Multi-Armed Bandits (MABs) for our search approach. A classic MAB problem is defined by a finite set of arms $A = \{a_1, a_2, \ldots, a_n\}$, where each a_i is an independent random variable with finite expected value μ_i. The index of the optimal arm is thus,

$$i^* = \arg\max_i \mu_i .$$

We consider the "simple regret" variant of MAB problems where the objective is to quickly identify an arm that has expected value close to that of a_{i^*}. This identification is done by "pulling" a sequence of arms, where pulling an arm a_i results in observing a sample from a_i. An MAB algorithm after n pulls maintains an estimate of what it considers to be the optimal arm i_n. The regret after n pulls is thus equal to:

$$\text{REGRET}_n = \mu_{i^*} - \mu_{i_n}$$

Typically, the design of MAB algorithms for simple regret aim to optimize the decrease of REGRET_n as a function of n.

In the context of our pipeline search, one could equate arms to pipelines and pulls to the evaluation of a pipeline, which is typically a random variable due to randomness in cross-validation runs. However, there are an enormous number of arms in this case, which requires leveraging structure in the problem that is not usually available in atomic MAB problems. In particular, our arms are highly structured and also have common structures among them. This led us to develop a two stage MAB approach for pipeline search.

Two-Stage MAB for Pipeline Search. As described earlier the arms, or pipelines, can be decomposed into structure and parameters. We thus, formulate this decomposed search approach as a two-stage MAB problem. The first stage MAB, called the Structure MAB, selects which pipeline structure should be improved upon next. The second stage MAB, the Component MAB,

then selects a ‘component’ within the selected structure. The component can be either a full vertex of the structure (i.e. a primitive) or an individual hyperparameter of a vertex/primitive. Once a component is selected, the approach attempts to find an improvement for that component (see below). This two-stage selection of a component followed by optimization of the component continues until time runs out. Pseudocode for this approach is in Figure 9, which we now describe in further detail.

```
function EveSeach
  Inputs:
      StructurePolicy - MAB policy for structure selection
      ComponentPolicy - MAB policy for component selection
      N - Number of pipelines to return
  Returns:
      List of pipelines

  History <- Empty list
  StructureArms <- List containing only [RANDOM_ARM]
  while termination conditions have not been met:
    SArm <- StructurePolicy(History, StructureArms)
    if SArm = RANDOM_ARM:
        NewPipelines <- GenerateRandomPipelines()
    else:
        ComponentArms <- GetComponents(SArm.PipelineStructure)
        Component <- ComponentPolicy(History, ComponentArms)
        BestPipeline <- FindBestPipelineWithStructure(History, SArm.PipelineStructure)
        NewPipelines <- ImprovementPolicy(History, BestPipeline, Component)
    Results <- EvaluatePipelines(NewPipelines)
    Update History with Results

  BestPipelines <- Top N entries in History with the highest validation score

  return BestPipelines
```

Figure 9: Pseudocode description of Eve's pipeline search.

To introduce new pipelines into the system, the first Structure MAB has a special ‘Random’ arm which forces the system to generate several random structures and evaluate those structures based on the default parameters of the corresponding primitives. The pipeline with the best performance among the generated ones is then returned as the result of pulling the random arm. The ‘Random’ arm thus allows the Structure MAB to include completely new pipelines rather than selecting among only existing structures. An MAB algorithm will tend to pull the ‘Random’ arm if generating random pipelines appears to be improving performance more compared to pulling arms for existing structures.

After an individual component is selected by pulling the Structure MAB followed by the Component MAB, the algorithm applies an *improvement operator* (via the call to **ImprovementPolicy**) to attempt to search for an improvement to the pipeline by modifying the selected component several times and evaluating the performance of the modified pipelines. We considered two different improvement operators:

- **Random Assignment** – randomly assign a value to the selected component based on the type of the component.
- **Bayesian Optimization (BO)** – a global-optimization technique that uses predictive modelling (typically via Gaussian Processes) to guide the selection of values to evaluate. A complete description of BO is beyond the scope of this report, but an excellent reference is (Snoek, Larochelle, & Adams, 2012). However, for those familiar with BO the details of our settings are as follows. For numeric components we use a standard BO approach with a Gaussian Process model and the Expected Improvement acquisition function. For discrete parameters we use simple empirical distribution models for each discrete value and the Expected Improvement acquisition function.

Figure 10 provides pseudocode for these two improvement operators.

```
function RandomImprovement
    Inputs:
      History - List of past pipeline evaluations
      Pipeline - Fully-configured pipeline to improve on
      Component - Component in the pipeline to focus on improving
    Returns:
      A list of pipelines to evaluate
  for i in [1..IMPROVEMENT_LENGTH]:
    Candidate <- Pipeline
    Candidate[Component] <- Random assignment of legal parameter values for Component
    return Candidate

function BOImprovement
    Inputs:
      History - List of past pipeline evaluations
      Pipeline - Fully-configured pipeline to improve on
      Component - Component in the pipeline to focus on improving
    Returns:
      A list of pipelines to evaluate
  for i in [1..IMPROVEMENT_LENGTH]:
    Candidate <- Pipeline
    RelevantHistory <- History filtered to only include pipelines in the same family as Pipeline
    Xs <- Parameter set for Component in RelevantHistory
    Ys <- ValidationScore for entries in RelevantHistory
    Model <- BuildGP(Xs, Ys) ;; builds a Gaussian Process model
    NextXs <- OptimizeEI(Model)
    Candidate[Component] <- NextXs
    return Candidate
```

Figure 10: Pseudocode describing several of the pipeline improvement policies that were evaluated.

MAB Algorithm Choice. To complete the specification of our approach we need to specify the MAB algorithm used to select among arms at each iteration. Our first approach was to select arms for the Structure MAB via the popular UCB algorithm (Auer et al., 2002). This algorithm maintains upper confidence bounds on each arm based on observations so far and always pull the arm with the highest upper confidence bound. We found, however, that this approach did not perform well in our setting and would often result in performance that was significantly worse than random. This is in part due to the fact that UCB is designed to optimize "cumulative regret" rather than "simple regret". That is, UCB tries to optimize the sum of returns of all arm pulls rather than just trying to identify the highest performing arm. This can cause UCB to be less exploratory, which apparently was detrimental in our setting. More importantly, however, the Structure MAB arms are not stationary distributions as assumed by UCB. That is, each pull of an arm corresponds to selecting a structure and improving that structure based on all prior information. This means that repeated pulls of a single structure arm will tend to result in an increasing performance profile for that arm, rather than i.i.d. observations from a stationary distribution.

The above observation led us to consider MAB algorithms for the "improving bandits" setting (Heidari, Kearns, & Roth, 2016), which are specifically design to handle bandits where the arms tend to improve with the number of pulls (and eventually have performance that levels off). At a high-level the arm pulling strategy we use for the improving bandits setting estimates the improvement slope of each arm and uses that to decide which arm to pull next based on the expected improvement. Figure 11 provides pseudocode for the three arm pulling strategies considered for our Structure MAB. This includes:

- **Random** – select an arm uniformly at random.
- e--**greedy** – with probability 1 - E select the best performing arm and otherwise select a random arm. This algorithm is known to have theoretical advantages over UCB for optimizing simple regret (Snoek et al, 2012).
- **Improving Bandit –** an implementation of the improvement-bandit MAB algorithm from (Heidari et al., 2016).

```
function RandomMABPolicy
    Inputs:
      History - List of past pipeline evaluations
      Arms - List of available arms to pull
    Returns:
      The arm to pull
  a <- RandomChoice(Arms)
  return a

function EGreedyMABPolicy
    Inputs:
      History - List of past pipeline evaluations
      Arms - List of available arms to pull
      Epsilon - Chance to behave greedily
    Returns:
      The arm to pull
  if Random() < Epsilon:
    a <- Arm in Arms with the best averaged reward according to History
  else:
    a <- RandomChoice(Arms)
  return a

function ImprovingBanditSlopeMABPolicy
    Inputs:
      History - List of past pipeline evaluations
      Arms - List of available arms to pull
    Returns:
      The arm to pull
  BestSlope <- 0
  BestArm <- null
  for each Arm in Arms:
    ThisStructure <- Arm.PipelineStructure
    MostRecent <- Latest performance of the structure ThisStructure in History
    LocalBestSlope <- 0
    for i in [1..MEMORY_LENGTH]:
      PastScore <- ith most recent performance of the structure ThisStructure in History
      LocalBestSlope <- Max(LocalBestSlope, (MostRecent - PastScore) / i)
    if LocalBestSlope > BestSlope:
      BestSlope <- LocalBestSlope
      BestArm <- Arm
  return Arm
```

Figure 11: Pseudocode describing several multi-arm bandit arm selection policies that were evaluated.

For the second-stage Component MAB problem we also considered this same set of bandit algorithms. This gave us nine possible configurations of the two-stage bandit selection process, each of which could use either one of the two improvement operators.

3.4.2.3 Extensions

Building on our framework, we extended it over the one used during the February evaluation. The new framework separates the search over data preparation from the search over machine learning (ML) pipelines, which can provide a better factoring of the search space. It also separates the search over the discrete pipeline schemas and the parameters for those discrete schemas. Since a critical aspect of data science is correct representation of the data, the data processing deserves its own separate search and optimization step. Meanwhile, separating the choice of primitives from hyperparameters significantly reduces the complexity of search.

We studied the issue of making search as effective as possible subject to time limits. We proposed the following two approaches:

- Take the time taken to evaluate a pipeline into consideration during search and choose pipelines that maximize the expected improvement per unit time.
- Reason about the time available when planning the search.

For example, the search algorithm might be faced with a choice between improving a current pipeline, which is fast but has a limited ceiling, versus exploring a different part of the search space, which is slower to yield improvements but has a higher ceiling. The second option would only be chosen if there is enough time to search the new part of the search space thoroughly enough to obtain improvements.

We also studied the issue of minimizing the impact of expensive pipeline evaluation during search. We explored the following two strategies:

- Only partially evaluate candidate pipelines at first and evaluate the most promising pipelines more fully
- Intelligent reuse computation between different evaluations in the same search process

Studying these two questions would have prepared us well for future phases where time and resource issues could become significant.

3.4.3 Integration of Eve Automated Model Composition Software

During Year 1 of the Eve project, we integrated the Eve automated model composition software by developing and refining APIs for integration, integrating Eve TA2 elements in preparation for integration events, and participating in semi-annual D3M integration events with TA1 and TA3 performers.

During the effort, we updated the API for the POE. The engine's API was designed to consume information from the other Eve components using the following methods:

- getProblem() returns the problem description from the User Manager
- getPrimitives(pquery) returns primitives that match *pquery* from the Primitives Manager
- putFeedback(p) provides feedback on *p*'s quality to the Primitives Manager
- definePipeline(p) allows POE to specify a pipeline and returns a unique pipeline key
- executePipeline(pk) informs the Execution Manager to execute pipeline *pk*
- rankPipeline(pk, i) ranks the pipeline *pk* as the *ith* pipeline
- getDataset(dquery) returns data sets that match *dquery* from the Datasets Manager
- putData(data) provides the key to the stored data to the Data Store
- getData(key) returns keyed data from the Data Store

The following methods get results from the POE:

- getPipelines() returns a ranked list of pipelines
- getSolution(pk) completes and returns the pk^{th} pipeline

We integrated the POE into Eve, and also implemented the common TA1 API for use within Eve. We updated the newest TA2-3 API and match the dynamic script to the new interface.

We redid hooks for the POE and prepared entry points for evaluation.

We converted all Scala portions of Eve to Python so it could better integrate with the Pipeline Optimization Engine. This new Python-based Eve system did not require reloading the primitive libraries before executing each pipeline. Therefore, the conversion to Python resulted in a much

faster pipeline executor. We integrated the new version of our system with the TA2-3 API used in the February evaluation, and reimplemented support for the API.

We ran tests of the Python-based Eve system on the seed problems and generated a table of results including reasons for failures and number of pipelines tested.

We performed a partial integration with the Purdue Modsquad TA3 system, and integrated Eve within TA3 Kubernetes configurations.

3.4.4 Evaluation

During Year 1 of the Eve project, we worked with SMEs and other D3M performers to exercise and evaluated the D3M toolkit's ability to build complex data modeling pipelines for user-specified data and outcomes of interest.

We evaluated Eve based on the three key dimensions in Table 3.

Table 3: Metric Dimensions

Dimension	Example Metrics
Accuracy	Performance of the models developed by Eve and the decisions/contributions made by SMEs. This is evaluated by directly contrasting SME inputs and results with Eve with inputs and results from working with data scientists. Specific targets include: plans, models, decisions, workflows, and problem formalizations.
Efficiency	Time and cost savings introduced by Eve. This includes analyzing the time commitment required by the SME, as well as the cost savings from removing the data scientist from the equation. This also includes efficiency assessments of the underlying process, as well as a comparison of the number of models Eve can explore.
Complexity	Complexity of the problems that Eve can address. This includes the type and specification of the problem (e.g., empirical science problems with complete data vs. underspecified or unresolved problems), as well as quantitative notions of complexity (e.g., number of primitives needed; amount of data; etc.).

4. Results

4.1 Internal Testing Results

In addition, to program evaluations, we tested our search and planning system internally. Interestingly, when considering different configurations, we found that there was no statistically significant difference between using Bayesian optimization (BO) as an improvement policy and using pure random sampling as an 'improvement' policy. This contradicts our prior experience in using BO for difficult optimization problems, where there is almost always a significant improvement over random. One hypothesis is that the parameter optimization problems generated here are densely populated with solutions that are close to optimal. While time did not allow for a detailed investigation into this, it does appear to be a reasonable explanation based on limited experiments, where we created dense plots of some of the response surfaces corresponding to our parameter optimization problems.

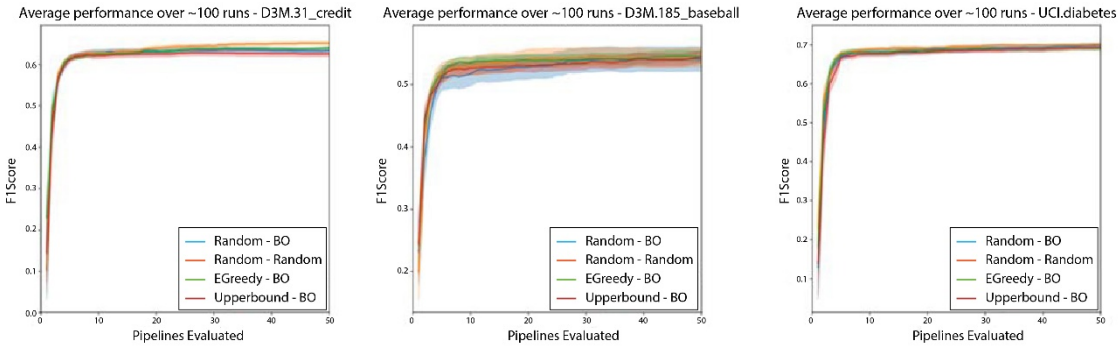

Figure 12: Plots of averaged performance vs number of pipelines evaluated for several combination of selection and improvement policies.

Our second surprise was that we were not able to see significant improvements over the random multi-armed bandit (MAB) strategy (at either stage) compared to the non-random strategies. The non-random strategies did not decrease performance compared to random, which was the case for UCB, but no consistent improvement over random was observed. Figure 12 shows the results of several of these experiments, demonstrating the near-identical performance of different combinations of MAB arm selection policies and improvement policies when averaged over many independent runs. This was very surprising and led us to question whether there was a fundamental flaw in the system.

To help assess this, we evaluated our “random bandit approach” (random arm selection policy, random hyperparameter improvement policy) against the state-of-the-art autoML system AutosSKLearn. We found that when we gave both systems the same amount of wall clock time or the same number of pipelines to evaluate they were able to find pipelines with similar performance.

Papers on AutoSKLearn and other autoML systems do not include comparisons to randomized discrepancy searches such as ours, which is, perhaps, one reason that this phenomenon has not been mentioned in the literature. An alternative explanation is that we were using a misconfigured version of AutoSKLearn. However, we are able to achieve results with our distribution of AutoSKLearn that are close to the results reported in the literature, suggesting the library was working as intended. We also spent considerable time ourselves optimizing the AutoSKLearn configuration.

Figure 13 shows an example comparison of running our approach “Eve” and AutoSKLearn (shown as “AutoML”) on 5 benchmark datasets for 300 seconds each. We see that Eve and AutoSKLearn are typically on par regardless of AutoSKLearn’s configuration. Interestingly we did not observe that Metalearning or Ensembling lead to a consistent impact on the performance of AutoSKLearn.

Dataset	Ensemble Size	Metalearning	Times (sec.)	Seed	Training Date Size	Testing Data Size	AutoMLF1 Score	Eve Test Score (Best Pipeline)	Eve Policy
delta_ ailerons	1	0	300	1	5346	1783	0.940301289	0.9414103575	Random
	50	0					0.9436103542		
	50	25					0.9436465695		
	1	25					0.9419866422		
diabetes	1	0	300	1	576	192	0.7557843731	0.7351956556	Random
	50	0					0.7597296959		
	50	25					0.7352860075		
	1	25					0.766750771		
spect	1	0	300	1	200	67	0.6954545455	0.6867695185	Random
	50	0					0.6894226717		
	50	25					0.6954545455		
	1	25					0.6259441708		
31_credit	1	0	300	1	750	250	0.6381578947	0.6954545455	Random
	50	0					0.6548232695		
	50	25					0.6773268321		
	1	25					0.6125927738		
seismic_ bumps	1	0	300	1	1938	646	0.4832	0.5621840986	Random
	50	0					0.4832		
	50	25					0.4998656997		
	1	25					0.5006763285		

Metalearning - Initialize the hyperparameter optimization algorithm with this many configurations which worked well on previously seen datasets. Disable if the hyperparameter optimization algorithm should start from scratch

Ensemble - Number of models added to the ensemble built by Ensemble selection from libraries of models. Models are drawn with replacement

Figure 13: Comparison of our system's performance "Eve" against AutoSKLearn (shown as "AutoML") with different configurations of AutoSKLearn.

For our internal evaluation, in the setting we were working in, we were essentially tied with our state-of-the-art, and had little reason or evidence to suggest that we could continue to improve our system's performance with the right selection or improvement policies. Our belief is that the differences in performance would be apparent with larger or more difficult datasets, or that a large collection of pipeline evaluation metadata could be used to effectively drive the MAB arm selection, but we no longer have the resources to pursue that line of research.

4.2 Program Evaluation

For the program evaluation, we participated in a D3M evaluation event in February 2018. The evaluation requirements for this event were not available prior to the evaluation, and there was no framework or code available to test against. We redid hooks for the POE and got entry points ready for the evaluation; integrated TA1-2 API hooks into Eve; and created a Docker container. We packaged Eve into the Docker container (a standalone executable package) for evaluation, and ran it according to specified commands from the Government evaluation document. We ran actual TA1 primitives during the evaluation and Eve components ran inside a TA3 setup. We also finalized the search procedure over pipelines for the evaluation event, which was based on a randomized Monte-Carlo search. We worked closely with the NIST team to identify issues that applied not only to our effort but also to other teams. In the February evaluation, we were able to automatically discover TA1 primitives based on their metadata, ingest TA3 problems and datasets, perform searches for pipelines, and execute those pipelines.

We also participated in an evaluation run by NIST in April 2018. We collaborated with other performers and government teams to understand and refine the APIs for this evaluation, and

continued updating Eve system components to meet these requirements. We prepared a new version of our TA2 system for the April evaluation.

We developed a set of sample primitives used in our pipeline execution engine and identified an optimizer (SMAC) that can generate pipeline descriptions using those primitives, and created a Docker image for evaluation. We made progress on integration among various Eve components—integrating Eve with TA1 and TA3 systems, and building a Docker component for evaluation. We participated in the 5-week integration event, continued to extend and integrate Eve components into TA1 and TA3 systems, and continued to build and evaluate the Docker container for Eve.

In the NIST evaluation, Eve was able to perform some of the evaluation tasks but was unable to perform many of them. We believe many of the difficulties resulted from the fact that we were automatically discovering TA1 primitives, rather than using our own primitives. Unfortunately, TA1 primitives were not yet available for some data types, so we were unable to find suitable primitives. For other TA1 primitives, their metadata was inadequately curated and sometimes incorrect. For example, hyperparameters that we believed to be in the legal range for the primitives caused the primitives to crash. This made it difficult for a fully automated approach to succeed.

5. Conclusions

We have developed Eve, a virtual data scientist to serve the role of TA2 in the D3M system. Eve receives problems and datasets from TA3, discovers TA1 primitives, searches for pipelines using those primitives to solve the problem, and provides the execution of those pipelines. Tests show that our approach has promise and demonstrates in proof of concept that the approach can work on some problems, but significant progress is needed to produce a full working system.

We have identified curation of TA1 primitives as a critical need going forward. It is necessary for us to understand what primitives do and how they can be used, and in particular what the valid hyperparameter ranges are for them. A well understood and more complete set of primitives will enable us to provide coverage for a wider range of problems in a more reliable manner.

6. References

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-Time Analysis of the Multi-Armed Bandit Problem. *Machine Learning, 47*, 235–256.

Balla, R. K., & Fern, A. (2009). UCT for Tactical Assault Planning in Real-Time Strategy Games. In *IJCAI* (pp. 40–45).

Brochu, E., Cora, V. M., & de Freitas, N. (2010). A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *ArXiv:1012.2599 [Cs]*. Retrieved from http://arxiv.org/abs/1012.2599

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., … Colton, S. (2012). A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games, 4*, 1–43.

Dietterich, T., Dereszynski, E., Hutchinson, R., & Sheldon D. (2012). Machine Learning for Computational Sustainability. Presented at the Green Computing Conference (IGCC), International, San Jose, CA. http://doi.org/DOC: 10.1109/IGCC.2012.6322258

Heidari, H., Kearns, M., & Roth, A. (2016). Tight Policy Regret Bounds for Improving and Decaying Bandits. (pp. 1562–1570). Presented at the IJCAI.

Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). Sequential Model-Based Optimization for General Algorithm Configuration (pp. 507–523). Presented at the International Conference on Learning and Intelligent Optimization.

King, B., Fern, A., & Hostetler, J. (2013). On Adversarial Policy Switching with Experiments in Real-Time Strategy Games. In *ICAPS*.

Lloyd, J. R., Duvenaud, D., Grosse, R., Tenenbaum, J. B., & Ghahramani, Z. (2014). Automatic Construction and Natural-Language Description of Nonparametric Regression Models. *ArXiv:1402.4304 [Cs, Stat]*. Retrieved from http://arxiv.org/abs/1402.4304

Mulwad, V., Finin, T., & Joshi, A. (2013). Semantic Message Passing for Generating Linked Data from Tables (pp. 363–378). Presented at the International Semantic Web Conference.

Rahm, E., & Do, H. H. (2000). Data Cleaning: Problems and Current Approaches. *IEEE Data Eng. Bull., 23*, 3–13.

Raman, V., & Hellerstein, J. M. (2001). Potter's wheel: An Interactive Data Cleaning System. In *VLDB* (Vol. 1).

Ramnandan, S. K., Mittal, A., Knoblock, C. A., & Szekely, P. (2015). Assigning Semantic Labels to Data Sources (pp. 403–417). Presented at the European Semantic Web Conference.

Ritze, D., Lehmberg, O., & Bizer, C. (2015). Matching HTML Tables to Dbpedia. In *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics* (p. 10).

Singh, R., Rocktaschel, T., Hewitt, L., Naradowsky, J., & Riedel, S. (2015). WOLFE: An NLP-Friendly Declarative Machine Learning Stack. In *Proceedings of NAACL-HLT* (pp. 61–65).

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms (pp. 2951–2959). Presented at the Advances in neural information processing systems.

Syed, Z., Finin, T., Mulwad, V., & Joshi, A. (2010). Exploiting a Web of Semantic Data for Interpreting Tables. In *Proceedings of the Second Web Science Conference* (p. 5).

Taheriyan, M., Knoblock, C. A., Szekely, P., & Ambite, J. L. (2016). Learning the Semantics of Structured Data Sources. *Web Semantics: Science, Services and Agents on the World Wide Web*, *37*, 152–169.

Thornton, C., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2013). Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In *in Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 847–855).

Wu, B., & Knoblock, C. A. (2016). Maximizing Correctness with Minimal User Effort to Learn Data Transformations. In *Proceedings of the 21st International Conference on Intelligent User Interfaces* (pp. 375–384).

Zumel, N., Mount, J., & Porzak, J. (2014). *Practical Data Science in R*. Shelter Island, NY: Manning.

7. List of Symbols, Abbreviations, and Acronyms

ACRONYM	Definition
API	Application Programming Interface
ARMA	Autoregressive Moving Average
CSV	Comma-Separated Values
DM	Datasets Manager
DS	Data Store
HMI	Human Machine Interface
JSON	JavaScript Object Notation
MARVIN	Modular Affective Reasoning-Based Versatile Introspective Architecture
ML	Machine Learning
NIST	National Institute of Standards and Technology
OSU	Oregon State University
PCA	Principal Component Analysis
PDL	Pipeline Description Language
PM	Primitives Manager
POE	Planning and Optimization Engine
REST	Representational State Transfer
RL	Reinforcement Learning
SMAC	Sequential Model-Based Algorithm Configuration
SME	Subject Matter Expert
SVM	Support Vector Machine
UM	User Manager