AFRL-RI-RS-TR-2021-131

SECOND GENERATION METALEARNING FOR DATA-DRIVEN DISCOVERY OF MODELS

BRIGHAM YOUNG UNIVERSITY

JULY 2021

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

AFRL-RI-RS-TR-2021-131 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
PETER J. ROCCI JR.
Work Unit Manager

/ S /
SCOTT D. PATRICK
Deputy Chief,
Intelligence 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

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
JULY 2021	FINAL TECHNICAL REPORT	MAR 2017 – MAR 2021

4. TITLE AND SUBTITLE

SECOND GENERATION METALEARNING FOR DATA-DRIVEN DISCOVERY OF MODELS

5a. CONTRACT NUMBER FA8750-17-2-0082

5b. GRANT NUMBER N/A

5c. PROGRAM ELEMENT NUMBER 63760E

6. AUTHOR(S)

Kevin Seppi
Brandon Schoenfeld

5d. PROJECT NUMBER D3ME

5e. TASK NUMBER 00

5f. WORK UNIT NUMBER 01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Brigham Young University
Research Administration Office
A285 ASB
Provo UT 84602-1231

8. PERFORMING ORGANIZATION REPORT NUMBER

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

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

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

11. SPONSOR/MONITOR'S REPORT NUMBER AFRL-RI-RS-TR-2021-131

12. DISTRIBUTION AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Our contributions to D3M center on our work on D3M framework. We (Brandon Schoenfeld) lead the Meta Learning working group. The group created the Pipeline and Pipeline Run Schemas and the associated framework. We contributed to the designs. We produced the Experimenter, Rerun, many test cases. We also created Primitives (TA1) for metafeature extraction, missing value imputation, and data profiling.

15. SUBJECT TERMS

Automatic Machine Learning, Metalearning, Deep Learning, Machine Learning Frameworks

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 J. ROCCI**
U	U	U	UU	36	**19b. TELEPHONE NUMBER** (*Include area code*) **N/A**

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

Table of Contents

1.0 Introduction 1

2.0 Summary 2

2.1 Definitions 3

3.0 Methods, Assumptions, and Procedures 5

3.1 D3M Framework Methods 5

3.2 Research Methods 7

3.2.1 Metalearning over Preprocessors 7

3.2.2 Metalearning in the Pipeline Space 9

4.0 Results and Discussion 16

4.1 D3M Metalearning Framework Contributions 16

4.1.1 Experimenter 16

4.1.2 Rerun 16

4.1.3 Primitive Contributions 17

4.2 Research Contributions 17

4.2.1 Metalearning over Preprocessors Results 18

4.2.2 Metalearning in the Pipeline Space Results 22

5.0 Conclusions 26

5.1 D3M Framework Contribution Conclusions 26

5.2 Research Contribution Conclusions 26

5.2.1 Metalearning over Preprocessors 26

5.2.2 Metalearning in the Pipeline Space 26

6.0 Recommendations 28

7.0 References 29

List of Figures

Figure 1: A Pipeline as a DAG 11
Figure 2: Dynamic Neural Architecture (DNA) 14
Figure 3: Relative Pipeline Runtime Above Baseline 19

List of Tables

Table 1: Number of Datasets where a Preprocessor Improved Performance 18
Table 2: Preprocessor Breakdown of Pipeline Accuracy Above Baseline 20
Table 3: Agent Comparison in AutoML Simulation 21
Table 4: Metamodel RMSE 23
Table 5: Metamodel Spearman 24
Table 6: Metamodel nDCG 25

1.0 Introduction

The demand for machine learning tools and data mining experts is growing faster than the supply. This gap is driving the demand for more automatic machine learning (AutoML) systems. AutoML systems can simplify the search for a machine learning solution and can even place machine learning tools into the hands of non-experts. The core component to such a system receives as input a dataset with a precise description of the problem or taskto be solved; as output, it returns a list of evaluated and ranked models for the user to consider as solutions to the input problem. The Data-Driven Discovery of Models (D3M) project provides solutions to this problem, but perhaps equally importantly a framework for the development of such solutions.

The framework for AutoML provided by D3M includes (among other things): the ability to run models in a common context, remember the performance obtained by those models, and compare the results obtained. Most of our D3M work contributed to that framework.

2.0 Summary

As noted in the introduction, the demand for machine learning tools and data mining experts is growing faster than the supply. Google's Prediction API, Amazon MachineLearning, DataRobot.com, and Microsoft's Azure Machine Learning are a few commercially available services that are attempting to fill this gap. Given the variety of datasets and problems, combined with the multitude of potential solutions or models, a primary concern for AutoML is to find a competitive solution in a reasonable time frame.

Existing AutoML systems utilizing tools like genetic search, Bayesian optimization, Monte-Carlo tree search, reinforcement learning, and metalearning. The Data-Driven Discovery of Models (D3M) project employs a metalearning-based approach. Metalearning seeks to learn to find good approaches for AutoML from past examples of ML problems, in the same way that regular ML seeks to learn from past examples of some application domain (identifying fraudulent business transactions, filtering out spam, or recognizing pictures of cats).

A key element of the D3M system is its formalization of machine learning solutions as "pipelines". These DAG-structured specifications, with the associated D3M reference pipeline execution tooling, supports ML algorithms from a wide range of diverse libraries, diverse data and problem types, a high degree of reproducibility, and full execution logging.

The D3M system includes:

- Pipeline schema - the model that allows D3M to fully specify how data is to be processed including which algorithms ("Primitive") will be use and in what combination
- Runtime - the system that executes a Pipeline instance

- Pipeline Run Schema - the model used to record (log) the execution and results of the execution of a Pipeline instance
- Evaluation Workflow - the system that evaluates an AutoML approach by running the pipelines it has generated and recording its performance
- Metalearning Database - a globally shared database for storing Pipelines and Pipeline Run instances

Our contributions to D3M center on our work on the framework described above. We note that Mitar Milutinovic is the primary architect of that framework, and likewise acknowledge the work of Diego Martinez, and the teams at NASA Jet Propulsion Laboratory (JPL), Data Machines (DMC), and other D3M performers. We (Brandon Schoenfeld) lead the Metalearning working group (which had primary responsibility for the framework), the D3M framework is a result of the working group as a team.

The specifics of our contributions include framework contributions, "primitive" contributions, and research contributions. All of these are described in the body of this report. In each of the sections which make up the body of this report you will find two subsections related to:

1. Framework (and primitives), and
2. Research contributions

2.1 Definitions

Machine learning datasets and problems are closely related. A dataset contains the variables (data) and a problem describes the nature of the mathematical model needed for a given dataset (i.e. which independent variables should be used to predict which response variables). The solution to a problem is a pipeline of machine learning algorithms or primitives. A primitive is a high-level machine learning algorithm or function which transforms data in some meaningful way. A primitive may have both parameters, which are set when the primitive is fitted on a set of data, and hyper parameters, which are set when the primitive is instantiated and govern the behavior of the primitive. Examples of such primitives range from classical algorithms implemented in standard libraries, like Scikit-learn [1], to the latest neural network structures. A pipeline is the functional composition of one or more primitives. While this definition of a pipeline is general, D3M focuses on pipelines that take as input a dataset and output predictions, or estimates of the response variable. A pipeline's performance is measured in terms of some function of the response variable and the predictions, like accuracy or root mean squared error. Note that the structure of the composition of primitives is often not strictly sequential, but rather can be represented by a directed acyclic graph (DAG),or computation graph, in general, e.g. in the case of an

ensemble. The term model can be applied to both a primitive or a pipeline, indicating that it is a stand-alone regression or classification algorithm (or composition of algorithms). The set parameters of a pipeline is union of the parameters of its constituent primitives and the hyper-parameters of a pipeline are defined similarly. Note that a library of primitives and a language for pipelines are existing products of the D3M program.

To evaluate a pipeline or model, one (pseudo-)randomly partitions the dataset into splits. Often there are three splits of a dataset: one for tuning the model parameters, one for tuning the model hyper-parameters, and one for selecting the model. When an AutoML system is searching for a solution to an input dataset and problem, this partitioning allows for testing the ability of the considered models to generalize to novel data.

Metalearning can be described as the application of machine learning tools to machine learning data, or metadata. A single instance of metadata consists of a dataset, a problem, a pipeline, a performance function, the value of the performance function when the pipelineis executed or run on the dataset and problem, and compute time. With enough such data points, one can assemble a metadataset, formalize a meta-problem, and apply metalearning as an intermediate step for an AutoML system.

Approved for Public Release; Distribution Unlimited.

3.0 Methods, Assumptions, and Procedures

Our Method's, Assumptions, and Procedures can be divided into 3 parts: 1) the D3M framework produced significantly by the metalearning working group, 2) Our work in preprocessor evaluation, and 3) our work in Pipeline performance modeling (DNA).

3.1 D3M Framework Methods

The D3M framework [2] is valuable as an execution environment for AutoML, but it also addresses other significant problems. It allows us to analyze the AutoML approaches and systems themselves, not just the ML algorithms. The complexity of such systems, the diversity of modern problem types, and the variety of ML algorithms all complicate the comparison of AutoML systems. For example, how can we compare an AutoML system thatsearches over neural networks with another that searches over graphical models? These two AutoML systems may differ in the resources required to run, the speed at which they can build an ML model, the type of data they use, and the search algorithm used to search the space of models.

The D3M framework formalizes ML programs precisely as end-to-end pipelines with reference pipeline execution tooling and supports ML algorithms from a wide range of diverse libraries, potentially raw data, diverse data and problem types, a high degree of reproducibility and execution logging, and the creation of a cross-system metalearning database. It provides a public metalearning database populated with executed pipeline information.

Currently, there exist many AutoML systems in both academia and industry, but to ad- vance the state of the art in AutoML, D3M required a increasingly sophisticated evaluation methods. The D3M framework formalizes ML programs precisely as end-to-end pipelines with reference pipeline execution tooling and supports ML algorithms from a wide rangeof diverse libraries, potentially raw data, diverse data and problem types, a high degreeof reproducibility and execution logging, and the creation of a cross-system metalearning database. It provides a public metalearning database populated with executed pipeline information. To enable evaluation of AutoML systems as described above and to enable a shared metalearning database between those AutoML systems, D3M includes a framework for auto-generated ML programs expressed as pipelines.

In the D3M framework, primitives can be written in any programming language, but they must expose Python interfaces that need to be extended from a set of base classes and additional mix-ins. Additionally, a primitive defines metadata describing itself, defines its parameters (state), which are usually learned from sample input and output data, defines hyper-parameters, which are general configuration parameters that do not change during the lifetime of a primitive, and defines types of inputs and outputs. There are two main types of hyper-parameters a primitive can use to define its hyper-parameters configuration. Tuning hyper-parameters potentially influence the predictive performance of the primitive, e.g.,

learning rate, depth of trees in a random forest, and architecture of the neural network. Control hyper-parameters control the behavior (logic) of primitives, e..g, whether or not a primitive is allowed to discard the unused columns.

The framework prescribes data types that can be passed between steps in a pipeline. Currently, a narrow set is allowed (Numpy ndarrays, Pandas DataFrames, Python lists, and Datasets). The Dataset data type serves as a starting point for a pipeline and can represent a wide range of input data, including raw data. Data types are extended to support the storage of additional metadata, for which the framework provides a standardized schema for many use cases. For example, semantic types are standardized descriptions of the meaning ofthe data, not just its representation in memory. The dataset data type is a unified represen- tation of the inputs that allow us to describe the relationships among multiple components and hints about how the data should be read. Some examples where the dataset repre- sentation is essential is when the dataset contains media (image, audio, video) distributed into multiple resources or datasets that are spread among multiple tables such as graph or relational data.

By design, the D3M framework allows for metalearning across AutoML systems because all pipelines share the same set of primitives and use the same language to describe the ML programs. All pipeline run documents can be contributed to a public metalearning database.

Existing work has attempted to address some of the best practices identified in Section 3. The AutoML Challenge Series (ChaLearn) [3] conducted evaluations by running system-submitted code on blind datasets. The AutoML Benchmark [4] attempts to identify which evaluation datasets were also used for system building, for example, which datasets Auto-sklearn used to build its internal metamodel. The datasets used in ChaLearn (6 rounds with 5 datasets each) represented a variety of tasks, data distributions, and metrics, but were preprocessed into a tabular format, potentially causing generated ML programs to be atypical. A third benchmarking study [5] comparing four popular systems noted that “[m]anyopen datasets require extensive preprocessing before use” and limited their study to clean OpenML [6] datasets. All of these benchmark evaluations compared systems that each used their own set of ML building blocks instead of a shared set, for practical reasons. However, conclusions about pipeline search and optimization strategies (e.g., is Bayesian optimization better than genetic optimization?) are limited because differences in system performance areconfounded by the differences in algorithm availability to each system. Using the pipelines generated in the above studies for automatic warm-starting or metalearning between systemsis not feasible because each system uses its own pipeline representation. There are some popular pipeline languages which might be candidates for such a purpose. Scikit-learn’s
[1] pipeline allows combining multiple scikit-learn transforms and estimators, supporting tabular and structured data, but not raw input files. Common Workflow Language [7] is a standard for describing data analysis workflows with a focus on reproducibility. However, it also focuses on combining command line programs into workflows, a pattern not generally followed by AutoML-made ML programs. Kubeflow (kub) simultaneously provides a pipeline language and simple Kubernetes deployments. It supports the combining of components that use different libraries, but every component is a Docker image, thus requiring inputs and outputs to be serialized instead of directly passing memory objects between components.

The D3M framework was designed to support evaluations based on the best practices outlined above. As an example, a third-party organization (Data Machines) evaluated eight AutoML systems built upon the framework. Blind datasets were used to prevent all sys- tems from over-fitting to known datasets. Evaluation results thus show how well systems generalize to new tasks in terms of both pipeline performance and the number of task types supported. All systems evaluated used the exact same set of ML building blocks, includ- ing the same versions. Therefore, differences in scores can be attributed to system designs. Researchers can isolate design differences and improve system strategies, thus advancing AutoML research. Note that these advances are limited to the types of ML building blocks given to systems. For example, the pipeline search strategies that produce the best pipelines with Scikit-learn [1] algorithms might be different from those strategies that produce thebest neural network pipelines. A similar caveat can be made for the types of tasks on which systems are evaluated. Still, evaluation of systems built using our framework allows us to explore these types of questions and test related hypotheses. An advantage of using this standardized ML framework across ML systems is that there are already tools available that wrap [8] AutoML systems to expose the same control interface to unify AutoML system execution and that facilitate analysis of AutoML system results [9, 10]. There is potential even to automate AutoML design itself via metalearning over the metalearning database.

3.2 Research Methods

3.2.1 Metalearning over Preprocessors

This subsection describes our work on metalearning for preprocessors. See Schoenfeld et al. [11] for more details.

Over the past couple of decades the work on automating the machine learning process has mainly focused on algorithm selection [12]. In recent years, this work has been extended to include the important issue of hyper-parameter optimization [13, 14]. Given that practical applications of machine learning often rely not only on algorithms that can be applied to data, but also on transforming the data prior to such application, the natural next stepis to consider the need to design and/or select pipelines consisting of both preprocessing algorithms and model building algorithms, as in [15].

In the context of AutoML, metalearning consists of using machine learning to determine solutions to new machine learning problems based on data from the application of machine learning to other problems. For each application of machine learning, we observe a dataset, an algorithm or sequence of algorithms, and some measure of performance. Together, these components represent one training data point for metalearning. With enough such data points, a metadataset can be assembled and machine learning be applied to it, with the goal of gaining insight into the machine learning process. We extend the work of metalearningto include the selection of a single preprocessor to improve the performance of a chosen classifier, thus taking a step toward an entire ML pipeline recommendation system.

Using data from over 10,000 machine learning experiments, our pre-processor work shows that preprocessing data before passing it to a classification algorithm tends to hurt classification accuracy on test data, but shortens both the average training runtime and average prediction runtime for the entire pipeline. At the same time, the most accurate pipelines (both on training and test data) within our experiments are far more likely to use somekind of preprocessing algorithm. These results suggest that when building machine learning pipelines, especially when doing so automatically, preprocessors should be used, but must bechosen carefully. We make use of metalearning on the data we collected to build predictive models of when a preprocessing algorithm will improve a particular classifier's accuracy or runtime. Our results suggest that metalearning can intelligently reduce the search space for AutoML systems by guiding the preprocessor selection process when building entire machinelearning pipelines.

Preprocessing Experiments We select 192 classification datasets from OpenML [6]. We use approximately a 70/30 train/test for each dataset, since cross-validation would result in prohibitive computational cost for data collection. Since many preprocessing and classification algorithms do not tolerate sparse or non-numeric data, especially as implemented in scikit-learn [1], we "clean" all datasets by first imputing missing values, and second one-hot encoding all categorical variables. Performing this cleaning on all datasets allows for controlled comparisons across experiments. Any differences in pipeline performance between two experiments cannot be attributed to this cleaning as it is held constant across all experiments.

Imputation is performed on each variable, independent of the others, and agnostic to the target class, by randomly selecting a known value from that same variable. This imputation method assumes that missing values are not a separate category of their own, but that they represent data that was either not measured, not recorded, or lost. This method has the advantage over imputing the mean or the mode in that it naturally tends to preserve the original distribution of data in each column.

We consider all possible pipelines of length 3 or 4, with the first 2 steps fixed to our data cleaning process, i.e., imputer and encoder, followed by 0 or 1 preprocessor, and finally 1 classifier, for a total of 10,368 pipelines. We select 8 preprocessing algorithms and 6 classification algorithms, all implemented in scikit-learn, as follows:

- Preprocessing Algorithms:
  - Min-Max Scaler (MMS)
  - Standard Scaler (SS)
  - Select Percentile (SP)
  - Principal Component Analysis (PCA)
  - Fast Independent Component Analysis (ICA)
  - Feature Agglomeration (FA)
  - Polynomial Features (PF), and

- Radial Basis Function Sampler (RBFS)

- Classification Algorithms:
  - Random Forest Classifier (RFC)
  - Logistic Regression (LR)
  - K-nearest Neighbors Classifier (KNN)
  - Perceptron (Per)
  - Support Vector Classifier (SVC), and
  - Gaussian Naive Bayes (GNB)

All algorithms use their default hyper-parameter settings. Since our interest lies in determining which preprocessing algorithms to use, if any, we setup, as a baseline comparison, the 1,152 pipelines of length 3, which do not use any preprocessing algorithm. We only compare pipelines of length 4, i.e. with a preprocessor, against the baseline pipeline run on the same dataset, such that the only difference between compared pipelines is whether a preprocessor is present.

3.2.2 Metalearning in the Pipeline Space

We developed a solution to the problem of estimating the performance of any pipeline (which may be a DAG) [16]. With this estimator, an AutoML system could quickly optimize over the pipeline search space, with respect to the estimated performance measurement. We concentrated this work on tabular classification datasets.

We took a step beyond previous work and attempted offline metalearning on ML pipelines by exploiting the metadata of pipelines with potentially arbitrary structure. Using machine learning, we should be able to extract any patterns in pipeline performance between pipelines that are composed of one or more of the same ML algorithms. We utilize the ML framework we helped to develop as part of the Data-Driven Discovery of Models (D3M) program [2] to create and execute pipelines, and thus build a meta-dataset. We then outline the metamodels we use based on how they operate on the metadata. For tractability, we scope this work to classification tasks of small to moderate size and create pipelines with only a few different structures; however, our approach is applicable to any task type and pipelines with arbitrary structure.

Metadata The D3M program gathered datasets for AutoML system development and testing. As part of that effort, MIT Lincoln Labs created annotations that describe dataset metadata (especially semantic column types), distinguish the features from the targets, identify metrics to optimize, and include a static train/test split. All of these components facilitate automatic pipeline synthesis. From these curated datasets, we selected 194 classification datasets, which were originally sourced from OpenML [6], UCI Machine Learning Repository [17, 18], and Zenodo [19]. These datasets were chosen because they were otherwise publicly

available and such that metafeatures could be computed within a few minutes per dataset. A complete list of the datasets we used can be found in the thesis.

We computed metafeatures using a package we implemented in Python to contain the union of metafeatures presented in an R package [20] and OpenML [6]. This tool computes simple, statistical, information-theoretic, landmarking, and model-based metafeatures. Examples of these include the number of instances in the dataset, the mean of each class probability, class entropy, 1-nearest neighbor error rate, and decision tree mean level size, respectively. We then selected the intersection of metafeatures that were computable across each of our selected datasets, resulting in 72 total metafeatures. A complete list of metafeatures can be found in the thesis.

We generated pipelines that would be representative of typical, human-made machine learning pipelines, including straight pipelines with a single classifier and ensembles of three classifiers. Generating pipelines instead of using results from OpenML, for example, allows us to explore DAG pipeline and enables us to perform a grid search over all possible ML algorithm combinations in a constrained pipeline space. The straight pipelines consist of a missing value imputation ML algorithm, a feature preprocessing algorithm (including the no-op algorithm), and a final classifier. The imputation ML algorithm sampled values from each column independently and uniformly over all instances with known values. We also create DAG pipelines that are ensembles of three straight pipelines, after the imputation step, and use a simple voting algorithm to make final classifications.

Figure 1 shows a template for straight and DAG pipelines. Two pipeline structures allows us to show a proof of concept for exploiting granular pipeline metadata that could be expanded to arbitrary DAG structures. We included 9 feature preprocessing algorithms (10 including the no-op), 14 classification ML algorithms, and 7 “glue” algorithms:

- Feature Preprocessing
  - FastICA
  - GenericUnivariateSelect
  - KernelPCA
  - MinMaxScaler
  - Nystroem
  - PCA
  - SelectFwe
  - SelectPercentile
  - StandardScaler
- Classification
  - BaggingClassifier
  - BernoulliNB

  - DecisionTreeClassifier
  - ExtraTreesClassifier
  - GaussianNB
  - GradientBoostingClassifier
  - KNeighborsClassifier
  - LinearDiscriminantAnalysis
  - LinearSVC, LogisticRegression
  - PassiveAggresiveClassifier
  - RandomForestClassifier
  - SGDClassifier
  - SVC

- Glue
  - ColumnParserPrimitive
  - ConstructPredictionsPrimitive
  - DatasetToDataFramePrimitive, EnsbleVoting
  - ExtractColumnsBySemanticTypesPrimitive
  - HorizontalConcatPrimitive
  - RenameDuplicateColumnsPrimitive

We used Scikit-learn [1] implementations for feature preprocessing and classification algorithms, with all ML algorithms being wrapped inside a common D3M API.

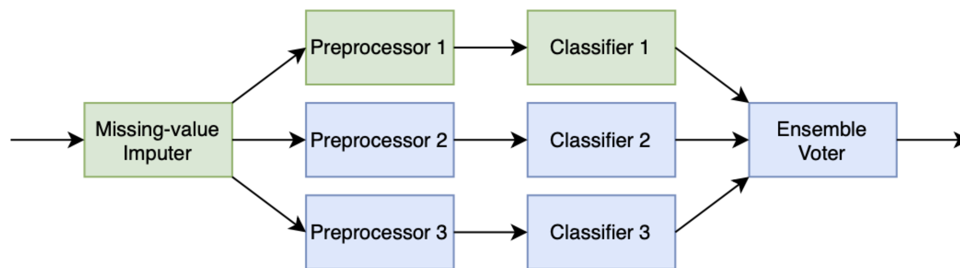

Figure 1: A Pipeline as a DAG

Figure 1: A pipeline can be represented as a directed acyclic graph (DAG). This example pipeline is a template for the pipelines generated in our metadata. A dataset is input on the left and data flows from left to right, following the arrows. Each node transforms the data with some ML algorithm and the final predictions are output on the right. The green nodes show the structure of a straight pipeline. The green and blue nodes combined show an

ensemble pipeline. “Glue” ML algorithms for parsing data, manipulating data structures, etc., are part of the actual pipelines, but are omitted here for expediency.

After removing pipelines that were non-functional on our collection of datasets (due to out-of-memory errors, ML algorithm bugs, or timeout errors), there remained 4,592 different pipelines. All ML algorithms use default hyper-parameters because exploring the hyper-parameter space of each pipeline is outside the scope of this work. We collected the results of 413,567 distinct runs of pipelines on our chosen datasets using a modest compute cluster with 20 identical machines, each with 4 Intel Core i7-4770K CPUs and 32 GB DIMM1 RAM.

The metadata contained in each of these pipeline runs constitutes a single instance in the metadataset. Each instance consists of the dataset identifier, the dataset metafeatures, a DAG representation of the pipeline (including ML algorithm identifiers and input/output connections), and the pipeline’s score on the test set of the indicated dataset. We used the macro F1 score, though any classification metric could be used. The test macro F1 score is thus the predictive target for our metalearning tasks.

We randomly partitioned our metadataset into train, validation, and test sets, grouped by the dataset such that each split contained datasets and corresponding runs of pipelinesnot found in the other two splits. The train set contains 125 datasets with 225,668 total metadata instances, the validation set contains 25 datasets with 90,131 instances, and the test set contains 44 datasets with the remaining 97,768 instances. We choose to use a meta- holdout set instead of using meta-cross-validation due to computational tractability in tuningthe metamodels’ hyper-parameters on the meta-validation set.

Metamodels We consider a broad range of metamodels, including naive baselines, state-of-the-art baselines, classic machine learning models, and modern deep learning models. This diversity allows us to evaluate how well we can estimate pipeline performance and how well this estimate allows us to rank pipelines. These metamodels are implemented using Scikit-lean [1] and PyTorch [21]. We outline and group the metamodels based on the amount of the pipeline metadata they use to make their predictions.

Metadata Agnostic Metamodels The most naive metamodel, the constant Mean model, uses the average performance of all pipelines in the training metadata as its estimate of performance. Since the Mean model is constant, it cannot provide a ranking of pipelines, so we instead utilize the Random model as the naive baseline when evaluating pipeline rank- ing performance. This model generates a random ranking of pipelines, independent of the metadata.

Pipeline Agnostic Metamodels The next metamodel we consider is the one used by Auto-sklearn [13] to warm-start its Bayesian optimization process. Given a dataset’s metafeatures, it finds the k-nearest datasets (where distance is defined as the L1 distance between dataset metafeatures) and recommends the best pipeline from each of those nearest datasets.

We label this metamodel k-ND. Note that this model only provides a ranking of pipelines, not estimates of performance. Also, this approach under-utilizes available metadata as it ig- nores any pipeline structure and discards most instances of metadata except for the one best pipeline for each dataset. It is thus limited to recommending only those pipelines that were the best pipeline for some dataset within the train split of the metadataset, which is a small fraction of available pipelines. We also present results for an adaptation of Auto-sklearn's k-ND, k-NN regression, for estimating pipeline performance directly. This adapts k-ND by weighting pipeline scores of the nearest datasets by the normalized inverse L1 distance. Thisapproach allows us to use all available metadata instances instead of discarding most of it.

Pipeline Structure Agnostic Metamodels We include two more baseline models that utilize the information about which ML algorithms are used in a given pipeline, in addition to the dataset metafeatures, for estimating pipeline performance. The pipeline representation is simplified from a DAG structure to a vector of indicator variables for each ML algorithm. We use Scikit-learn's Linear Regression and Random Forest as a metamodels.

Instead of hand-picking more classical models, we bootstrap model selection by passing our meta-dataset to Auto-sklearn. We give this system the default 24 hours to generate and optimize an ensemble of Scikit-learn models. The resulting model is a metamodel, which we include in our evaluation and label Meta Auto-sklearn.

Linearized Pipeline DAG Metamodels Increasing the amount of metadata available, we explore a couple of deep learning sequence models: the LSTM [22] model and the attention-based Transformer [23] model (which we label Transformer). These metamod-els create a latent representation of pipelines by first requiring the DAG structure to be linearized into a sequence. The LSTM metamodel thus ignores some of the computational structure of the pipeline. The Transformer metamodel assumes that each step of the pipeline is dependent on all previous steps, according to the linearized order. While this loses some information about the computational structure of the pipeline, this preserves more pipeline information than any of the above approaches discussed. The encoding of the pipeline cre- ated by these metamodels is then concatenated with the dataset metafeatures and passed toa final set of fully connected layers for final pipeline performance estimation. We note that the Transformer model could be adapted to use a masking mechanism that would preserve the DAG structure of the pipeline, but leave this for future work.

Complete Metadata Metamodels The first of our final two models is the DAG LSTM [22, 24], which creates a latent embedding of the full pipeline DAG, without discarding structural information. The DAG LSTM has a single LSTM cell that passes over each node in the DAG. When a node has multiple outgoing edges, a copy of the current LSTM cell output and hidden state is made for each edge and the LSTM cell operates in parallel onthe subsequent paths in the DAG. When a node has multiple incoming edges, the outputs and hidden states from all incoming nodes are aggregated together (we use the element-wisemax operator) and passed to the LSTM cell. The final output of the final LSTM cell isthe pipeline embedding.

This embedding is concatenated with the dataset metafeatures and passed to a set of fully-connected layers for pipeline performance estimation. The loss is back-propagated through the full connected layers and the DAG LSTM.

The final model is our proposed Dynamic Neural Architecture, or DNA. In the context of visual question answering, a similar model was proposed as the Neural Module Network [25]. Unlike other models presented here, DNA does not create an embedding of the pipeline DAG. Its weights are not even explicitly trained on the input DAG, but only implicitly. DNA is initialized with three types of neural network modules: node modules, an input module, and an output module. In general, each module can be any neural network operator, but we use only blocks of fully-connected layers. A node module is instantiated for each type of node in the heterogeneous DAGs to be modeled, which in our case is one node module for each ML algorithm used to construct the pipelines.

At runtime, DNA's network is constructed dynamically by composing the network modules based on the input DAG, as shown in Figure 2. The network begins with the input module and ends with the output module. In the middle, the network is dynamically composed of the node modules that correspond to the nodes in the DAG and they are composed in the exact same graphical structure as the input DAG. The input to the network, in our case, is the dataset metafeature vector and the output is an estimate of pipeline performance. The dynamic composition of neural network modules requires that we fix a latent dimensionality between all the modules for interoperability. The input module enables us to map from the input dimensionality to the latent dimensionality; similarly, the output module maps from the latent dimensionality to the output dimensionality. When a module has multiple inputs from other modules, the inputs are aggregated together (again, we use the element-wise max operator), as in the DAG LSTM.

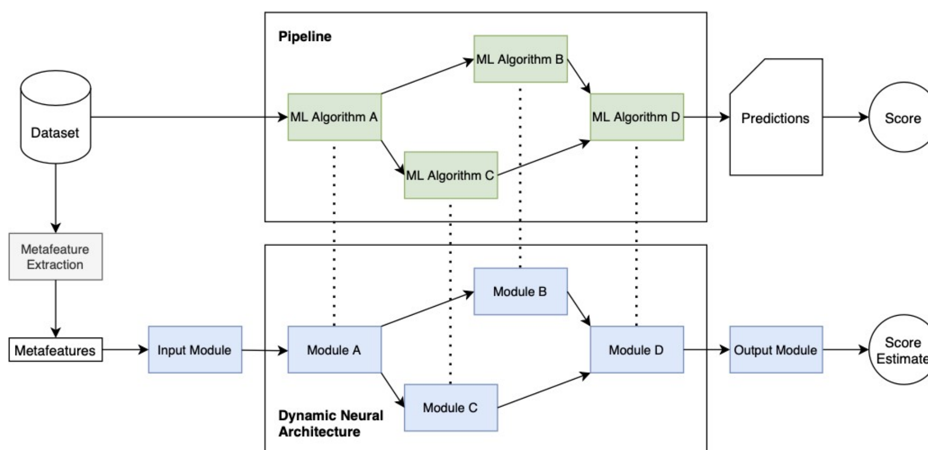

Figure 2: Dynamic Neural Architecture (DNA)

Figure 2: The Dynamic Neural Architecture (DNA) contains a distinct neural network module instance (e.g. a block of fully-connected layers) reserved for each type of node (e.g. ML algorithm) in the directed acyclic graphs (DAGs) (e.g. ML pipelines) to model. The neural

network is built dynamically at runtime by composing the reserved modules that correspond to each node in a given DAG in the same graphical structure as the DAG.

The DNA network can be trained using any typical loss function and optimizer. The net- work modules are trained jointly, though only those modules which are used in any given, dynamically assembled network are trained. Since the other modules did not make up the network, their weights are not updated. Because of this dynamic nature, only instances of data that contain the same DAGs can be (mini-)batched together.

4.0 Results and Discussion

4.1 D3M Metalearning Framework Contributions

As noted above we (Brandon Schoenfeld) led the Meta Learning working group helped define the pipeline schema, pipeline run schema and database. Our contributions as part of that committee included:

Pipeline schema: contributed to the design, wrote the JSON schema, and did some validation

Pipeline run schema: contributed to the design, wrote the JSON schema, and implemented the Python interface

Runtime: implemented pipeline-run portion, pipeline rerun, some CLI portions, majority of unit tests, refactoring (along with Mitar, Diego)

Evaluation Workflow: contributed to the design and wrote the document

Database: consistency script, documentation, and the vast majority of the data (as of the end of our participation in D3M)

Our contributions also include:

4.1.1 Experimenter

Our "Experimenter" is a system for generating a spectrum of Pipelines and corresponding Pipeline Runs. Note that an Experimenter is not directly attempting to be an AutoML system, rather it populates the Metalearning Database with a spectrum of Pipelines and Pipeline Runs. This is intended to provide (or augment) the training data needed for metalearning-based AutoML solutions. Our "experimenter", generated 94.4% of pipeline runs as of May 4, 2021.

4.1.2 Rerun

D3M pipelines contain all that is needed to run a Machine learning experiment. The Rerun tool accesses a specific pipeline and re-runs it, which is useful for re-evaluating a specific Run.

4.1.3 Primitive Contributions

We also contributed the following Primitives:

Metafeature Extraction Primitive The premise of metalearning is that pipelines which were successful in one circumstance (data, problems, etc.) will also be successful in other similar circumstances. In order to do so in must be possible to recognize when data is similar. The just as ML often employs feature extraction in simplify the space in which the ML system must learn, the Metafeature Extraction Primitive simplifies (characterizes) the space of data in which a metalearning approaches will learn.

Missing Value Imputer AutoML and ML in general often has to work in the context of imperfect data, often including missing data. We provided a simple primitive for plausibly filling in missing values.

Profiler Experiments with the D3M AutoML systems (TA2 and TA3's) revealed that human-provided (data) annotations lead to better AutoML results. Our profiler automati- cally generates similar annotations.

4.2 Research Contributions

In addition we have sought to advance metalearning in the context of our work on D3M, Specifically we have:

- **Preprocessor selection** Much of the work in metalearning has focused on classifier selection, combined more recently with hyper-parameter optimization, with little concern for data preprocessing. In our work, we conduct an extensive empirical study over a wide range of learning algorithms and preprocessors, and use metalearning to determine when one should make use of preprocessors in ML pipeline design.
- **Modeling Pipeline Performance** If one could model machine learning pipeline viability, that is, predict the performance of a given pipeline on a given dataset, AutoML would be significantly easier, since Pipelines could be evaluated without actually needing to run them. Even an approximate evaluation would be helpful in leading an AutoML system away from pipelines which are bad and toward ones that are at least likely to be performant. We proposed a dynamic neural architecture (DNA) metalearner to mimic the structure of any arbitrary pipeline and model its performance.

4.2.1 Metalearning over Preprocessors Results

Of the 10,368 possible pipelines we generated for our pipeline experiments, 10,331 ran to completion. Thirty pipelines failed due to a known, but yet unresolved, bug in the eigenvalue decomposition algorithm used by ICA and another one because ICA did not converge with default hyper-parameters. The final six failed on one dataset because the combination of one-hot encoding and the PF preprocessor caused the dimensionality of the feature spaceto explode, and all the classifiers ran out of memory. We chose not to adjust the hyper-parameters or modify the dataset to maintain comparability with the other pipelines.

An unexpected and counter-intuitive result is that adding a preprocessing step in a pipeline can actually reduce a pipeline's total runtime. 79.1% of the 9,179 non-baseline pipelines were trained in less time than their baseline. Similarly, when making predictions on test data, 80.2% of the non-baseline pipelines run in less time.

We observe in Table 1 that preprocessors which reduce the number of features in the dataset reduce runtime more often than those which create or add new features. For example, SP, which reduces the dimensionality by 90%, and FA, which keeps only two features, almost always reduce training time, while PF, which adds a quadratic number of features, more often increases training time. Since the algorithmic complexity of the classifiers often depends on the number of features, feature reduction will reduce runtime for the classifier. Based on these results, it appears that the cost of training a preprocessor is more than paid by the reduction in classifier training time.

Table 1: The comma separated values in each cell reflect the number of datasets for which the preprocessor improved train time, test accuracy, and both, in order, compared to the baseline pipeline. Preprocessors are enumerated along the rows and classifiers along the columns.

Table 1: Number of Datasets where a Preprocessor Improved Performance

	RFC	LR	KNN	Per	SVC	GNB	Mean
MMS	192,**23**,23	192,**43**,43	191,**84**,83	191,**104**,104	159,**77**,68	110,**42**,27	173,**62**,58
SS	192,**58**,58	181,**89**,84	188,**98**,96	192,**117**,117	138,**93**,71	101,**81**,41	165,**89**,78
SP	192,**27**,27	192,**45**,45	192,**65**,65	192,**56**,56	176,**60**,55	152,**84**,71	183,**56**,53
PCA	173,**59**,58	172,**58**,50	192,**30**,30	192,**88**,88	155,**4**,4	90,**106**,52	162,**57**,47
ICA	161,**38**,38	186,**56**,56	187,**77**,77	186,**83**,83	110,**31**,20	99,**75**,39	155,**60**,52
FA	191,**13**,13	192,**30**,30	192,**25**,25	191,**27**,26	165,**33**,31	155,**64**,52	181,**32**,30
PF	140,**97**,71	42,**96**,13	77,**49**,21	84,**79**,29	32,**45**,17	9,**71**,5	64,**72**,26
RBFS	119,**23**,17	172,**32**,24	165,**21**,18	181,**59**,53	86,**22**,7	41,**66**,11	127,**37**,22
Mean	170,**42**,38	166,**56**,43	173,**56**,52	176,**76**,70	128,**45**,34	95,**73**,37	151,**58**,46

Figure 3 shows the relative training time of pipelines, compared to their baseline. Interestingly, 25% are at least 67.9% faster than baseline. While most of the mass in the plots of Figure 3 lies below zero, there is a long tail with 3 pipelines taking over 1,000 times longer to train than their baseline because of the presence of the PF preprocessor.

Unlike runtime, preprocessing data tends to hinder pipeline accuracy. Of the 9,179 non-baseline pipelines, 69.4% yielded lower test accuracy than the baseline. However, considering the most accurate of all 10,331 pipelines for each dataset, we find that 91.1% do use a preprocessor. Similarly, for pipelines with accuracy within 5% and 10% of the top, 85.7% and 86.4%, respectively, do use a preprocessor. Table 1 shows that SS improves test accuracyfor more pipelines than any other preprocessor. Furthermore, Table 2 shows that SS isthe only preprocessor which improves both pipeline train and test accuracy on average. Perhaps classifiers can optimize over the feature space better when the data is zero-centered with unit variance. A similar argument could be made for MMS, which is the only other preprocessor to show any improvement in test accuracy. However, it might be the case that these preprocessors are simply well-suited to the datasets we considered.

Given the above results, one may wonder if it is possible to get both improved accuracy and runtime or if there exists a natural trade-off between the two. Thus, we turn our attention to the combined effect of accuracy on runtime, and vice-versa. By focusing on the high end of each spectrum, we observe the following.

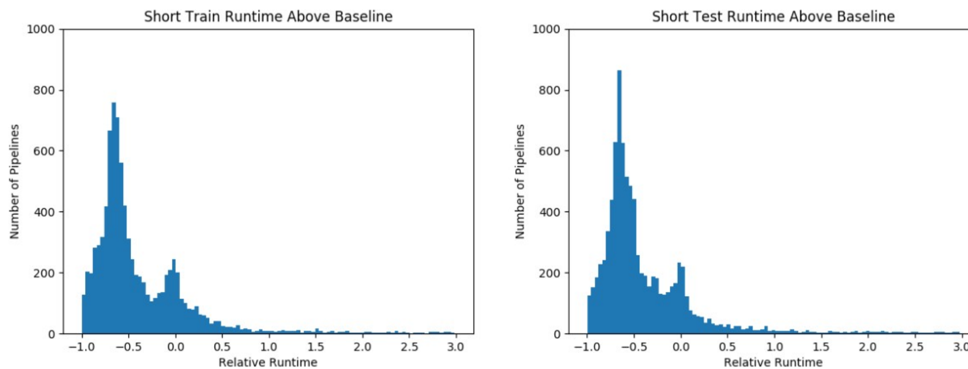

Figure 3: Relative Pipeline Runtime Above Baseline

Figure 3: Relative Pipeline Runtime Above Baseline. Both graphs have long tails which we have truncated at 3.0. The train graph (left) displays 8,764 of the 9,719 non-baseline pipelines; the maximum relative runtime is 2,024. The test graph (right) displays 8,709 pipelines, with a maximum relative runtime of 1,041. A value of 1.0 indicates that a pipeline took 100% more time to train or test than the baseline, while -0.5 means a pipeline took 50% less time.

Table 2: Preprocessor Breakdown of Pipeline Accuracy Above Baseline. In each cell, the first number is for training data and the second one for test data.

Table 2: Preprocessor Breakdown of Pipeline Accuracy Above Baseline

Preprocessor	MMS	SS	SP	PCA
Mean	-0.3%, 0.8%	2.3%, 2.3%	-10.0%, -5.71%	0.2%, -0.4%
StdDev	14.2%, 15.8%	13.1%, 15.7%	19.4%, 18.9%	11.8%, 12.4%

Preprocessor	ICA	FA	PF	RBFS
Mean	-6.2%, -6.2%	-15.9%, -15.5%	0.7%, -1.9%	-8.3%, -15.8%
StdDev	24.6%, 22.2%	23.5%, 25.7%	13.1%, 13.6%	24.2%, 24.9%

- For the 25% fastest training non-baseline pipelines, the average accuracy is 9.2% lower than would be obtained without the preprocessor.
- For the 25% fastest testing non-baseline pipelines, the average accuracy is 7.8% lower than would be obtained without the preprocessor.
- For the 25% most accurate non-baseline pipelines, the average training time is 126.6% higher, and the average test time is 93.1% higher, than would be obtained without the preprocessor.

On average, it appears that there is indeed no free lunch: a preprocessor will increase accuracy at the cost of runtime, or it will reduce runtime at the cost of accuracy. It thus becomes important to be able to intelligently select preprocessors so as to reduce the likelihood of choosing the wrong preprocessor.

We turn our attention to metalearning to further investigate the details of the above relationships, and thus inform the AutoML process. The predictive features of our metadatasetconsist of metafeatures extracted from the base dataset and a one-hot encoded preprocessor identifier. We extract 18 simple, 8 statistical, 1 information-theoretic, and 14 landmarking metafeatures, from each of the 192 base datasets. Our metafeature extraction tool is an extension of an existing R script [20]. The target class is a binary value indicating whether or not the given preprocessor produces a pipeline with test accuracy greater than or equalto the corresponding baseline pipeline. We use the random forest classifier (RFC) as our metalearner, as it showed best empirical performance on the meta experiments. The accuracy of the metamodel at predicting whether a given preprocessor (passed as input as part of the predictive features) will improve accuracy over not using some preprocessor is 62.6% on the test data. In contrast, if we instead merely select the mode class (which happens to be always choosing that the preprocessor will not help the classifier) we obtain 53.0

The value of metalearning in the context of AutoML systems is found when trained metamodels can be used to design better pipelines than alternative strategies. Here, we simulate an AutoML scenario in which a system must decide which preprocessor to use, given a par-

ticular task dataset and given that it has already chosen a classifier. To assess the added value of our metamodel, we create four agents which make this decision, one of which uses the RFC metamodel. Their strategies are:

1. None Agent. Choose never to preprocess. This represents much of the state-of-the-art, where only a classifier is selected.
2. Random Agent. Choose a preprocessor, or none, uniformly at random. This represents the case where the agent has no special prior knowledge of the preprocessors.
3. Mode Agent. Choose the preprocessor that is empirically the best, for each base classifier, based on past experiments. A veteran data scientist might begin here.
4. Oracle Agent. From the subset of preprocessors that the RFC metamodel predicts would be good, choose the one that is empirically the best based on past experiments.

The AutoML simulator splits the entire metadataset, of all experiment results generated earlier, into a 70/30 train/test split. Each agent is allowed to use the training portion of the metadata to inform their decisions on the test portion, though the None and Random agents do not. At test time, each agent is given a particular dataset and classifier and must decide which preprocessor to use, if any. The Oracle agent induces a metamodel for each base classifier. It queries the appropriate metamodel for each of the eight preprocessors. Of the preprocessors predicted to help, it selects the one that helped most often in the training metadata. If none of the preprocessors were predicted to help, none is chosen. Note that the Mode agent is a metalearning agent which takes full advantage of the training metadata, giving it a tremendous advantage over the None and Random agents. We measure the performance of each agent by computing what percent worse each agent is than an Optimal Agent who knows exactly which preprocessor (or none) to use, on each test task. This is only achieved by brute force search: running all possible pipelines. We aggregate these results across all test tasks, as shown in Table 3. The two metalearning agents, Mode and Oracle, perform best.

Table 3: Agent Comparison in AutoML Simulation. How much worse the agent-chosen pipelines performed relative to the best known pipelines for the test task.

Table 3: Agent Comparison in AutoML Simulation

Agent	None	Random	Mode	Oracle	Optimal
Mean	-10.51%	-15.40%	-6.57%	-6.19%	0
StdDev	17.23%	21.41%	12.23%	12.25%	0

4.2.2 Metalearning in the Pipeline Space Results

We tuned each metamodel's hyper-parameters using a random search over 70+ different hyper-parameter configurations on the validation split of the metadataset. We report final metamodel scores on the test split of the metadataset using the best hyper-parameter configuration found on the validation split. We combined the train and validation splits of metadata to train the metamodels for the final run on the test split. We ran the metamodels that used any type of pseudo-randomness five times and report the average and standard deviation over those runs. Since this work is separated from the context in which it would live, i.e., inside an AutoML system, we will explore these results from multiple points of view to illuminate the strengths and weaknesses of these different metamodels.

Estimating Pipeline Performance We evaluate the ability of each metamodel to estimate pipeline performance by measuring both the average spread of the errors with root mean squared error (RMSE) and the Pearson correlation between the predicted values and true values. The results are shown in Table 4. In general, there is an apparent trend that utilizing the granular pipeline metadata is associated with lower error and higher Pearson correlation when estimating pipeline performance.

The DAG LSTM has both the lowest RMSE and the highest Pearson correlation, perhaps because this metamodel utilizes the available metadata completely. Its weights are trained directly on the pipeline metadata. Furthermore, its DAG nature allows it to learn the relationships between the structure of the pipeline and the ML algorithms used in the pipeline. These results are more justified when comparing the DAG LSTM to the next-best models. The Transformer only uses linearized DAG information and the weights of the DNA model were not trained explicitly on the pipeline metadata. Despite the DNA's implicit use of the pipeline metadata, it is a competitive metamodel. One might then expect the LSTM to perform similarly to the Transformer, since they used the same representation of the meta- data. Perhaps the rigid, sequential nature of the LSTM is more limited in its ability to find patterns in the metadata than the more flexible attention mechanism of the Transformer.

Ranking Pipelines Next, we explore how well the metamodels' estimates of performance can be used to rank pipelines. In an applied AutoML context, a metamodel could estimate the performance of a set of pipelines on a given query dataset, and then use those estimates

Table 4: Mean and standard deviation of metamodel RMSE. RMSE (lower is better) and Pearson correlation (higher is better) over five runs with different random initializations. Scores in bold are within one standard deviation of the best mean score.

Table 4: Metamodel RMSE

Regression Model	RMSE	Pearson Correlation
Mean	0.302 ± 0.000	N/A
Linear	0.243 ± 0.000	0.680 ± 0.000
k-NN Regression	0.265 ± 0.000	0.524 ± 0.000
Random Forest	0.211 ± 0.001	0.739 ± 0.004
Meta Auto-sklearn	0.213 ± 0.001	0.739 ± 0.001
LSTM	0.376 ± 0.130	0.531 ± 0.184
Transformer	0.193 ± 0.005	0.782 ± 0.010
DAG LSTM	$\mathbf{0.182} \pm 0.006$	$\mathbf{0.799} \pm 0.010$
DNA	0.198 ± 0.005	0.758 ± 0.010

of performance to rank the pipelines. Because computational resources are always limited, this approach would allow the AutoML system to prioritize which pipelines to explore next.

Ranking pipelines by estimating performance has an advantage over directly learning to rank a set of pipelines. Ranking pipelines directly requires the metamodel to operate simultaneously on all candidate pipelines. Adding a single pipeline to that ranking requires re-ranking all pipelines. With an estimate of performance, adding a new pipeline to the ranking re- quires that the metamodel process only the new pipeline instead of all previously considered pipelines. This advantage applies to AutoML systems like Auto-sklearn [13], TPOT [26], and AlphaD3M [27, 28], where pipelines are generated dynamically at runtime.

We measure the quality of rankings made by the metamodels using the Spearman correlation coefficient (following Brazdil et al. [12]), as well as the normalized discounted cumulative gain (nDCG). The Spearman correlation measures the monotonicity of the estimated ranking with the true ranking and can only be used when a total ranking is provided. The nDCG metric is often used in information retrieval and combines the true pipeline score with the estimated ranking. Recall that the Mean and k-ND metamodels are unable to provide total rankings for a set of pipelines on a given dataset. The ranking results of our metamodels are shown in Table 5.

The metamodels that gave the best estimates of performance (Table 4) do not necessarily yield the best ranking in terms of Spearman correlation. This may suggest that there is too much error in the estimates of performance, which creates noisier rankings. Meta Auto-sklearn, though only moderately competitive in estimating pipeline performance, is significantly better at ranking pipelines than any other metamodel. While nDCG had the desirable property of incorporating the pipelines' performance with the ranking, it did not prove to be a very discriminatory measurement for choosing a metamodel.

Table 5: Mean and standard deviation of metamodel Spearman correlation. Spearman correlation (higher is better) and nDCG (higher is better) over five runs with different random initializations. Scores in bold are within one standard deviation of the best mean score.

Table 5: Metamodel Spearman

Model	Spearman Correlation	nDCG
Random	0.000 ± 0.006	0.948 ± 0.001
Linear Regression	0.437 ± 0.000	0.981 ± 0.000
k-ND (Auto-sklearn)	N/A	0.949 ± 0.000
Random Forest	0.538 ± 0.008	**0.983** ± 0.003
Meta Auto-sklearn	**0.566** ± 0.003	**0.984** ± 0.000
LSTM	0.320 ± 0.144	**0.973** ± 0.014
Transformer	0.543 ± 0.015	**0.986** ± 0.003
DAG LSTM	0.527 ± 0.013	**0.981** ± 0.008
DNA	0.521 ± 0.011	0.972 ± 0.006

Ranking at k: One may well argue that the quality of the total ranking of pipelines is much less relevant than the quality of the top few ranked pipelines. When we search the Internet, we do not care about the quality of the millions of search results, rather just thetop several results. We thus refine our evaluation of the metamodels to consider the quality of the top k ranked pipelines. We measure the nDCG@k, or the nDCG of the top k ranked pipelines. We also measure how many of the best pipelines were among the top k ranked pipelines and label it nBest@k. This gives us an idea of how many of the best pipelines are ranked highly.

One may also argue that ranking is not as important as the quality of the best pipeline. An AutoML system would likely run as many pipeline as possible until the time budget is spent, and then recommend the executed pipelines in the ordering of their true performance. In this scenario, the value of k would be determined dynamically. Thus, the quality of the underlying system design (i.e. metamodel) could be measured by the difference in performance between the best known pipeline and the best pipeline among the k executed pipelines. We call this measurement Regret@k. The results of the ranking at k measures are shown in Table 6, with k values of 25 (top) and 100 (bottom).

Interestingly, these results differ from the estimating performance and total ranking results. Most notably, the k-ND and Random metamodels have the lowest Regret@k with low standard deviation, despite not performing as well in the performance estimation and ranking measurements. This suggests that an AutoML system that uses the k-ND metamodel or simply random search of previously generated pipelines would effectively and reliably generate a good pipeline. However, this generalization may be limited to small pipeline spaces.

Regret@k is fundamentally different from the other measures in that it is not a type of averaging over many pipeline scores. A high nDCG@k or nBest@k indicates that the metamodel can select many good pipelines in the set of k. Given the noisy estimates of performance, perhaps there is a trade-off in the metamodels' ability to select one of the best pipelines and to select a pool of many good (but not quite the best) pipelines, among the k recommended.

Table 6: Mean and standard deviation of metamodel nDCG@, nBest@k, and Regret@k. nDCG@k (higher is better), nBest@k (higher is better), and Regret@k (lower is better) over five runs with different random initializations with k values of 25 (top) and 100 (bottom). Scores in bold are within one standard deviation of the best mean score.

Table 6: Metamodel nDCG

Model	nDCG@25	nBest@25	Regret@25
Random	0.759 ± 0.005	1.96 ± 0.16	0.014 ± 0.002
Linear Regression	0.886 ± 0.000	2.86 ± 0.00	0.024 ± 0.000
k-ND	0.805 ± 0.000	1.91 ± 0.00	**0.011** ± 0.000
Random Forest	**0.906** ± 0.007	**4.50** ± 0.53	0.026 ± 0.002
Meta Auto-sklearn	**0.900** ± 0.001	**4.09** ± 0.15	0.029 ± 0.001
LSTM	0.834 ± 0.059	2.95 ± 0.77	**0.045** ± 0.040
Transformer	**0.899** ± 0.007	**3.88** ± 0.62	0.044 ± 0.005
DAG LSTM	**0.890** ± 0.017	3.78 ± 0.29	0.042 ± 0.018
DNA	0.872 ± 0.013	3.83 ± 0.42	0.046 ± 0.012
Model	nDCG@100	nBest@100	Regret@100
Random	0.775 ± 0.004	16.7 ± 0.4	0.005 ± 0.001
Linear Regression	0.898 ± 0.000	22.1 ± 0.0	0.018 ± 0.000
k-ND	0.793 ± 0.000	17.0 ± 0.0	**0.002** ± 0.000
Random Forest	**0.916** ± 0.007	**28.8** ± 2.6	0.018 ± 0.002
Meta Auto-sklearn	**0.914** ± 0.000	25.6 ± 0.8	0.020 ± 0.001
LSTM	0.843 ± 0.056	20.3 ± 3.5	0.021 ± 0.017
Transformer	**0.912** ± 0.006	**26.6** ± 3.3	0.028 ± 0.004
DAG LSTM	**0.902** ± 0.018	26.0 ± 0.8	0.027 ± 0.013
DNA	0.886 ± 0.009	26.0 ± 1.1	0.022 ± 0.005

5.0 Conclusions

5.1 D3M Framework Contribution Conclusions

In this work we identified important best practices for AutoML systems. The D3M framework allows systems that are built on it to be evaluated according to these practices. The use of this framework by eight AutoML systems and their evaluation results presented help demonstrate the framework's viability. We observed that the framework can describe a diverse set of ML programs solving many ML task types. More important than the actual results obtained from any one particular evaluation is the ability to identify strengths and weaknesses specific to AutoML system designs and make improvements.

5.2 Research Contribution Conclusions

5.2.1 Metalearning over Preprocessors

We have analyzed the impact of selecting from among eight preprocessors in the context of six classification learning algorithms for the design of short ML pipelines. Our results suggest that preprocessing does not always improve accuracy, but in many cases does reduce training and prediction time. Using metalearning, we have shown that a metamodel can help AutoML systems determine more intelligently which preprocessor, if any, might help improve a pipeline's performance. Choosing the best metamodel or even meta-pipeline is left to future work. Future work also includes exploring longer pipelines, including hyper- parameter selection, and agents considering pipeline execution times.

5.2.2 Metalearning in the Pipeline Space

We conducted an offline metalearning experiment to estimate the performance of ML pipelines on novel tasks. Our approach expands on past metalearning work by exploiting granular pipeline metadata. In particular, the metamodels observe which ML algorithms are used at each step in the pipeline as well as the DAG structure of the pipeline. This approach advances beyond treating pipelines as entirely distinct black-box models and restricting the pipeline space. We then used this estimate of performance to rank pipelines as a hypotheticalpractical application of metalearning in an AutoML system.

We found empirically that the metamodels that utilize more granular metadata tend to estimate pipeline performance better. Our proposed Dynamic Neural Architecture (DNA) was competitive with the best metamodels, despite learning only implicitly from the pipeline metadata. On an absolute scale, the metamodels did not estimate performance very well, creating relatively weak rankings of pipelines. Selecting pipelines based on dataset similarity or even randomly ranking pipelines (that are already known to run) far outperform the more sophisticated metamodels.

This work is preliminary and calls for further metalearning research. Improved metafeatures could yield better representations of datasets, especially embeddings such as Dataset2Vec [29]. Learned embeddings would allow for end-to-end differentiable metalearning, such that metamodels could operate directly on raw datasets and pipelines to estimate pipeline performance. Only two pipeline DAG structures and default hyper-parameters and were used to limit the scope of the metadata generated. However, some of the metamodels we explored would be able to estimate the performance of pipelines structures not found in the training data. Combining DNA with a DAG embedding model, such as the DAG LSTM [30, 24], would perhaps perform better than either model alone. We could also expand the metamodels to operate on the hyper-parameters used, especially DNA because it creates a separate network module for each ML algorithm modeled. In short, richer metadata representations and much more metadata could be utilized to improve metalearning.

6.0 Recommendations

The D3M project covers many aspects of AutoML, however as implied by the name of the project, Data-Driven Discover of Models, the availability of data is a key element of the program. Thus we believe that the population of the metalearning database needs to be further supported and encouraged. We built the "experimenter", described above, but more needs to be done. One possible step in this direction would be to simplify the uploading and loading of data. Another possible improvement would be to run existing pipelines on other similar datasets (this would require both a new "experimenter" and a run service (described below).

More data could also be obtained if a broader community used the databases. Merging the metalearning database and schemas with OpenML would be one way to do that.

Similarly, data is not useful unless it can be accessed and provided in a convenient form. Specifically a tool is needed to convert the database into a metalearning dataset. We wrote a basic database download script (https://gitlab.com/datadrivendiscovery/metalearning-dataset) using the ElasticSearch API to download all documents, but should use the dumps. A big challenge is working with large amounts of data, especially as the database grows our script does not satisfy this need.

Furthermore, data needs to be valid and consistent in order to be useful. A Pipeline running service could address these needs. The existence of such a service would also provide a substantial motivation for using the D3M tools.

As noted earlier in this report the D3M framework could be applied to the AutoML problem itself, that is "Auto AutoML". A TA2 could be pointed at a metalearning dataset and used to update the TA2's internal model thus yielding life-long learning. A spin-off program would be needed to fully build that kind of metalearning-based AutoML.

Approved for Public Release; Distribution Unlimited.

7.0 References

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blon- del, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[2] M. Milutinovic, *Towards Automatic Machine Learning Pipeline Design*. University of California, Berkeley, 2019.

[3] I. Guyon, L. Sun-Hosoya, M. Boullé, H. J. Escalante, S. Escalera, Z. Liu, D. Jajetic, B. Ray, M. Saeed, M. Sebag, *et al.*, "Analysis of the automl challenge series," *Automated Machine Learning*, p. 177, 2019.

[4] P. Gijsbers, E. LeDell, J. Thomas, S. Poirier, B. Bischl, and J. Vanschoren, "An open source automl benchmark," *arXiv preprint arXiv:1907.00909*, 2019.

[5] A. Balaji and A. Allen, "Benchmarking automatic machine learning frameworks," *arXiv preprint arXiv:1808.06492*, 2018.

[6] J. Vanschoren, J. N. Van Rijn, B. Bischl, and L. Torgo, "Openml: networked science in machine learning," *ACM SIGKDD Explorations Newsletter*, vol. 15, no. 2, pp. 49–60, 2014.

[7] P. Amstutz, M. R. Crusoe, N. Tijanić, B. Chapman, J. Chilton, M. Heuer, A. Kartashov, D. Leehr, H. Ménager, M. Nedeljkovich, *et al.*, "Common workflow language, v1. 0," 2016.

[8] V. D'Orazio, J. Honaker, R. Prasady, and M. Shoemate, "Modeling and forecasting armed conflict: Automl with human-guided machine learning," in *2019 IEEE Interna- tional Conference on Big Data (Big Data)*, pp. 4714–4723, IEEE, 2019.

[9] J. P. Ono, S. Castelo, R. Lopez, E. Bertini, J. Freire, and C. Silva, "Pipelineprofiler: A visual analytics tool for the exploration of automl pipelines," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 390–400, 2020.

[10] C. A. Mattmann, S. Shah, and B. Wilson, "Marvin: An open machine learning corpus and environment for automated machine learning primitive annotation and execution," *arXiv preprint arXiv:1808.03753*, 2018.

[11] B. Schoenfeld, C. Giraud-Carrier, M. Poggemann, J. Christensen, and K. Seppi, "Pre-processor selection for machine learning pipelines," *arXiv preprint arXiv:1810.09942*, 2018.

[12] P. Brazdil, C. G. Carrier, C. Soares, and R. Vilalta, *Metalearning: Applications to data mining*. Springer Science & Business Media, 2008.

[13] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter,"Efficient and robust automated machine learning, 2015," *URL http://papers. nips. cc/paper/5872-efficient-and-robust-automated-machine-learning*.

[14] L. Kotthoff, C. Thornton, 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*, vol. 18, no. 25, pp. 1–5, 2017.

[15] P. Gijsbers, J. Vanschoren, and R. Olson, "Layered tpot: speeding up tree-based pipeline optimization," in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, September 18–22, 2017, Skopje, Macedonia*, pp. 49–68, CEUR-WS. org, 2017.

[16] B. J. Schoenfeld, "Metalearning by exploiting granular machine learning pipeline metadata," 2020.

[17] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository," 2017.

[18] M. Lichman, "UCI machine learning repository," 2013.

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

[20] M. Reif, "A comprehensive dataset for evaluating approaches of various meta-learning tasks.," in *ICPRAM (1)*, pp. 273–276, 2012.

[21] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, pp. 8026–8037, 2019.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L . Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[24] X. Zhu, P. Sobhani, and H. Guo, "Dag-structured long short-term memory for semantic compositionality," in *Proceedings of the 2016 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pp. 917–926, 2016.

[25] J. Andreas, M. Rohrbach, T. Darrell, and D. Klein, "Neural module networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 39–48, 2016.

[26] R. S. Olson and J. H. Moore, "Tpot: A tree-based pipeline optimization tool for au-tomating machine learning," in *Workshop on automatic machine learning*, pp. 66–74, PMLR, 2016.

[27] I. Drori, Y. Krishnamurthy, R. Rampin, R. Lourenço, J. One, K. Cho, C. Silva, and J. Freire, "Alphad3m: Machine learning pipeline synthesis," in *AutoML Workshop at ICML*, 2018.

Approved for Public Release; Distribution Unlimited.

[28] I. Drori, Y. Krishnamurthy, R. Lourenco, R. Rampin, K. Cho, C. Silva, and J. Freire, "Automatic machine learning by pipeline synthesis using model-based reinforcement learning and a grammar," *arXiv preprint arXiv:1905.10345*, 2019.

[29] H. S. Jomaa, L. Schmidt-Thieme, and J. Grabocka, "Dataset2vec: Learning dataset meta-features," *Data Mining and Knowledge Discovery*, vol. 35, no. 3, pp. 964–985, 2021.

[30] L. Song, Y. Zhang, Z. Wang, and D. Gildea, "N-ary relation extraction using graph state lstm," *arXiv preprint arXiv:1808.09101*, 2018.