Metalearning by Exploiting Granular Machine Learning Pipeline Metadata

Brandon J. Schoenfeld
Brigham Young University

Metalearning by Exploiting Granular Machine Learning Pipeline Metadata

Brandon J. Schoenfeld

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

ABSTRACT

Metalearning by Exploiting Granular Machine Learning Pipeline Metadata

Brandon J. Schoenfeld
Department of Computer Science, BYU
Master of Science

Automatic machine learning (AutoML) systems have been shown to perform better when they use metamodels trained offline. Existing offline metalearning approaches treat ML models as black boxes. However, modern ML models often compose multiple ML algorithms into ML pipelines. We expand previous metalearning work on estimating the performance and ranking of ML models by exploiting the metadata about which ML algorithms are used in a given pipeline. We propose a dynamically assembled neural network with the potential to model arbitrary DAG structures. We compare our proposed metamodel against reasonable baselines that exploit varying amounts of pipeline metadata, including metamodels used in existing AutoML systems. We observe that metamodels that fully exploit pipeline metadata are better estimators of pipeline performance. We also find that ranking pipelines based on dataset metafeature similarity outperforms ranking based on performance estimates.

Keywords: metalearning, automl, ml pipeline, metamodel, meta-dataset, metafeatures

ACKNOWLEDGMENTS

First and foremost, I thank God, our Father in Heaven, for His infinite love in my life. I thank my wife, Audrey, and my children, Oliver, Elsie, and Chloe, for their love and their constant support of my academic pursuits.

This work would not have happened without my first adviser, Christophe Giraud-Carrier, who is taking a three-year hiatus from his career to preside over the France Lyon Mission. I thank Kevin Seppi for stepping up to take over an entire research project, a group of students he did not know, and a research grant with many demands. David Wingate and Mark Clement have given valuable input as they have served on my Master's committee. Jennifer Bonnett, the Graduate Academic Advisor, and Erin Rowan, the Department Secretary, cannot be thanked enough for taking care of all the administrative details.

We gratefully acknowledge support from the Defense Advanced Research Projects Agency (DARPA), under award FA8750-17-2-0082, for our participation in the Data-Driven Discovery of Models (D3M) program. As part of the D3M program, many people contributed tools and infrastructure used in this work, especially Mitar Milutinovic, Diego Martínez-Garcia, Sujen Shah, Alice Yepremyan, and Shelly Stanley. I also want to thank Wade Shen, Brian Sandberg, and Joshua Elliott for their leadership in D3M.

I have worked with many other students who have contributed to the ideas, plans, code, and writing presented here. Thank you to Roland Laboulaye, Casey Davis, Mason Poggeman, Seth Donaldson, Jarom Christensen, Sean George, Ellie Van de Graffe, Orion Weller, Erik Huckvale, and Evan Peterson.

I will never be able to thank sufficiently those who motivated me to pursue my education. My mother, Celisa, sacrificed all her time, resources, comforts, and dreams to love and care for me and my brother, Mikael. Many others made my higher education (among many other things) possible, including Owen and Lynne Hamp, Rick and Alicia Loveridge, and Sylvia Danielson. I would not be here without you, thank you.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Automatic machine learning (AutoML) systems can use metalearning, either partially or entirely, to aid in machine learning (ML) program search. Such metalearning can democratize ML by aiding users in selecting suitable ML models, as well as facilitate learning from past results when presented with a new task [4]. One usable form of metalearning is to estimate ML program generalization performance, e.g. accuracy. Good estimates of ML model accuracy would allow an AutoML system to prioritize candidate ML program execution, favoring early execution of high performing ML programs before finite computational resources, especially time, are spent.

Modern ML programs contain multiple ML algorithms (e.g. data encoding, feature selection, missing value imputation, classification, ensembling, etc.) composed into an ML ***pipeline***. In general, the computational structure of a pipeline can be represented by a directed acyclic graph (DAG) with contextual, heterogeneous nodes (e.g. Figure 3.1). Each node in the pipeline corresponds to a stand-alone ML algorithm, with a different node type for each ML algorithm. Each node or ML algorithm has the additional context of the hyper-parameters used.

Related metalearning work (Chapter 2) has moved beyond metalearning over single algorithms to entire ML pipelines, but with some limitations. Often, pipelines are treated as black-box, stand-alone ML programs, even though they are composed of some of the same building blocks in many cases. This approach helps simplify an already challenging problem but restricts the metamodels' ability to learn from commonalities across pipelines. Other work has explored approaches that utilize the granular metadata of which ML algorithms comprise the pipelines, but do so in a restricted pipeline space, e.g straight pipelines. Simplifying assumptions preclude the possibility of metalearning over more interesting pipelines, such as ensembles.

We address the challenge of extending metalearning work to pipelines with potentially unbounded size and shape. This approach has immediate applicability in a system like TPOT [35], which uses a genetic algorithm to search for arbitrary pipelines, by speeding up pipeline search with estimates of performance. We explore a broad range of types of metamodels that utilize the pipeline metadata in varying degrees of granularity (Section 3.2), from naive metamodels to sophisticated deep learning metamodels, to estimate ML pipeline performance on novel tasks. We also propose a neural network architecture with the potential to model arbitrary DAG structures, which we call a Dynamic Neural Architecture (DNA). A similar model [1] was proposed in the context of visual question answering.

This work focuses on ***offline metalearning***, where the metamodels are trained to estimate pipeline performance on a novel dataset of interest given metadata about pipeline executions from a collection of other datasets. In contrast, ***online metalearning*** creates metamodels from metadata of pipelines executed only on the dataset of interest. We further evaluate our metamodels by considering how well their estimates of pipeline performance allow them to correctly rank pipelines. To our knowledge, this is the first work that presents metalearning models capable of handling arbitrary pipeline structures.

Chapter 2

Related Work

Metalearning can be used to predict the best algorithm (or pipeline) of a set, rank algorithms, or estimate algorithm performance [4]. Each of these can be used in offline, online, or hybrid approaches. However, all existing approaches have made simplifying assumptions about the pipeline search space to facilitate metalearning. Our work expands on existing work by removing the simplifying assumptions in the pipeline search space. For a more thorough metalearning survey see [51].

Offline metalearning approaches attempt to mimic human behavior by learning from past experience or results. Observing which ML algorithms, hyper-parameter configurations, or pipelines worked well on past tasks should help guide model search for novel tasks. A common metalearning approach is to extract metafeatures from datasets (thereby distilling a dataset down to a single vector representation) and then recommend the ML algorithms that performed best on datasets with similar metafeatures. An effort to provide a benchmarking dataset for metalearning [41] collected many datasets, ran several classification algorithms on those datasets, computed metafeatures for each dataset, and made the metadataset and the metafeature extraction tool publicly available.

Others have made similar meta-datasets and computed metafeatures to aid in selecting single classification algorithms [5, 12, 56], choosing initial hyper-parameter configurations in a fixed hyper-parameter space [19, 54], or selecting single preprocessing algorithms [44]. OpenML [52] hosts a large collection of metadata for offline metalearning, but their pipelines (flows) are currently limited to straight pipelines. Others have attempted to standardize metafeatures [43], explore which OpenML metafeatures have the most predictive power [3], use different distance metrics [33], automatically develop new sets of metafeatures [38], learn dataset distance with a Random Forest model [46], or follow the recent trends of using neural network embeddings to create Dataset2Vec [29]. Our work does not propose advances in dataset metafeature or similarity measures, but uses existing metafeatures.

There exist many styles of online metalearning. Various techniques exist for tuning hyper-parameters, such as random search [2], Bayesian optimization [47], and gradient-based methods [32]. Sequential Model-based Optimization (SMBO) [27] uses a surrogate model and an acquisition function to select the next hyper-parameter configuration for a given algorithm. AlphaD3M [15, 16] uses online deep reinforcement learning to explore the pipeline space. Internally, it uses an LSTM [26] to estimate the performance of straight pipelines and recommend actions to change the pipeline. TPOT [35] and AutoStacker [8] take an online evolutionary approach to modify pipelines, composed of preprocessing and classification algorithms. A more recent iteration of TPOT, Layered TPOT [24], estimates pipeline performance online by executing the pipeline on a subset of the data.

Hybrid approaches combine both offline and online metalearning approaches. Systems like Auto-sklearn [18] warm-start pipeline search with an offline approach, k-nearest datasets in metafeature space, followed by an online Bayesian optimization process. Furthermore, Auto-sklearn and AutoWeka [30, 50] use a hierarchical method that handles conditional hyper-parameters, treating the particular algorithm as the top-most hyper-parameter. The Mosaic system [40] implements an offline metalearning approach similar to Auto-sklearn but uses surrogate models for online metalearning.

Instead of combining separate offline and online techniques, hybrid approaches can also, for example, combine multiple past hyper-parameter optimization runs with a novel one into a single meta-loss function [20]. Probabilistic Matrix Factorization has been used to [22] estimate the performance of pipelines of length two, after a few pipelines have been executed to obtain a few data points online. This approach treats pipelines as indivisible entities, not compositions of stand-alone ML algorithms. The work we present would be immediately applicable to any AutoML system that uses an offline or hybrid offline-online metalearning approach.

Chapter 3

Methodology

We take a step beyond previous work and attempt 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 [34] 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.

3.1 Metadata

The D3M program gathered datasets[1] 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 [52], UCI Machine Learning Repository [13, 31], and Zenodo [25]. These datasets were chosen because they were otherwise publicly available and

[1]Available D3M datasets are found at `https://datasets.datadrivendiscovery.org/d3m/datasets`.

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

We computed metafeatures using a package[2] we implemented in Python to contain the union of metafeatures presented in an R package [41] and OpenML [52]. 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 Appendix B.

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 3.1 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 [28], GenericUnivariateSelect, KernelPCA [45], MinMaxScaler, Nystroem [14], PCA [55], SelectFwe, SelectPercentile, StandardScaler

[2]Our metafeature extraction tool can be found at https://github.com/byu-dml/metalearn or on the Python Package Index at https://pypi.org/project/metalearn/.

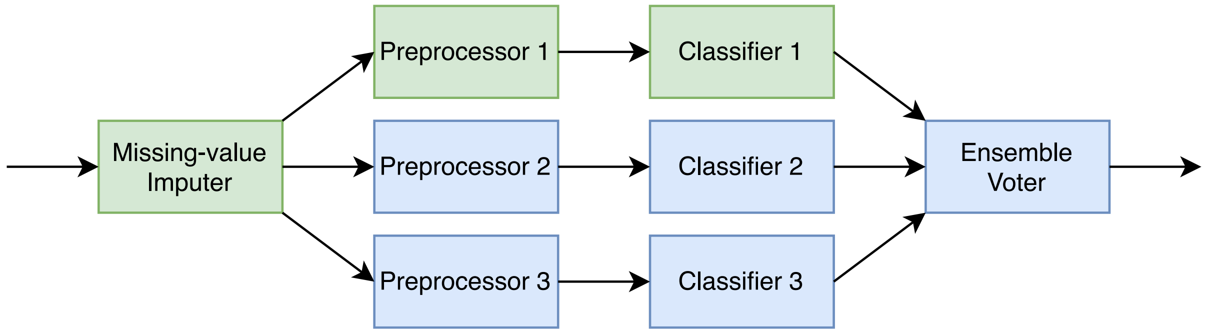

Figure 3.1: A pipeline can be represented as a directed acyclic graph (DAG). This example pipeline is a template for the pipelines generated in our metadata. A dataset is input on the left and data flows from left to right, following the arrows. Each node transforms the data with some ML algorithm and 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.

- Classification: BaggingClassifier [6], BernoulliNB [17], DecisionTreeClassifier [49], ExtraTreesClassifier [23], GaussianNB [17], GradientBoostingClassifier [21], KNeighborsClassifier [9], LinearDiscriminantAnalysis [42], LinearSVC [39], LogisticRegression [57], PassiveAggresiveClassifier [10], RandomForestClassifier [7], SGDClassifier [39], SVC [39]
- Glue: ColumnParserPrimitive, ConstructPredictionsPrimitive, DatasetToDataFramePrimitive, EnsembleVoting, ExtractColumnsBySemanticTypesPrimitive, HorizontalConcatPrimitive, RenameDuplicateColumnsPrimitive

We used Scikit-learn [37] implementations for feature preprocessing and classification algorithms, with all ML algorithms being wrapped inside a common D3M API.[3]

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.

[3]The D3M API is located at `https://gitlab.com/datadrivendiscovery/d3m`. ML algorithm annotations with links to the source code are at `https://gitlab.com/datadrivendiscovery/primitives`.

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 pipelines not 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 tuning the metamodels' hyper-parameters on the meta-validation set.

3.2 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 [37] and PyTorch [36]. 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 ranking 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 [18] 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 ignores 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. This approach 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*** [26] model and the attention-based Transformer [53] model (which we label ***Transformer***). These metamodels 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 created by these metamodels is then concatenated with the dataset metafeatures and passed to a 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*** [48, 58], 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 on the 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-wise max operator) and passed to the LSTM cell. The final output of the final LSTM cell is the 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 [1]. 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 3.2. 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.

The DNA network can be trained using any typical loss function and optimizer. The network modules are trained jointly, though only those modules which are used in any given,

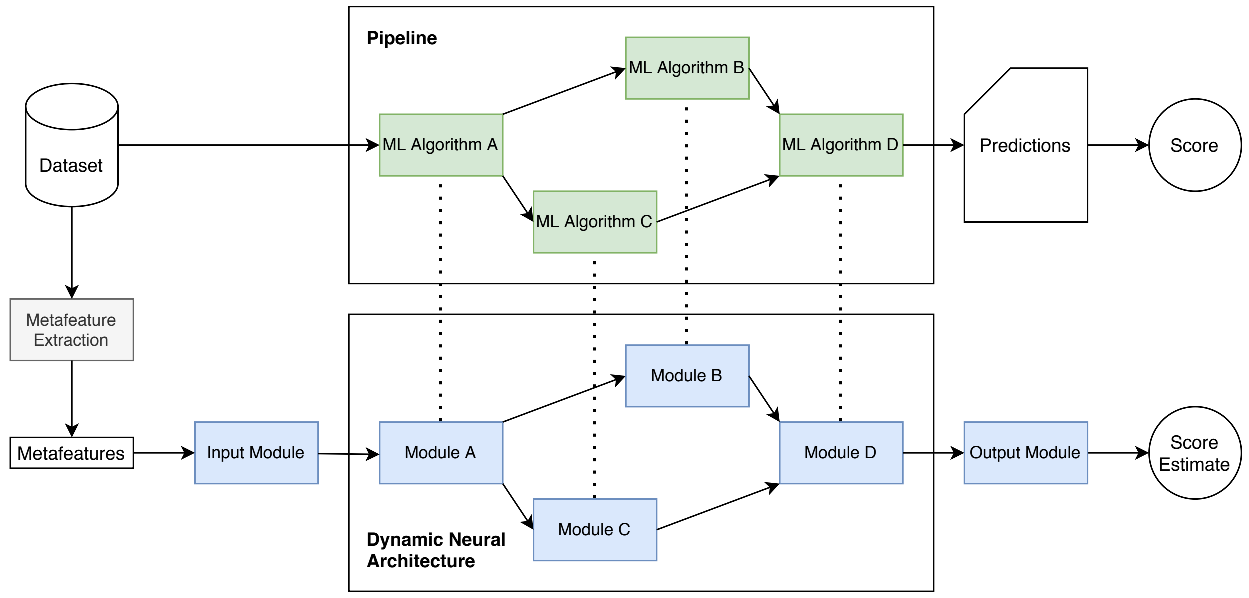

Figure 3.2: The Dynamic Neural Architecture (DNA) contains a distinct neural network module instance (e.g. a block of fully-connected layers) reserved for each type of node (e.g. ML algorithm) in the directed acyclic graphs (DAGs) (e.g. ML pipelines) to model. The neural network is built dynamically at runtime by composing the reserved modules that correspond to each node in a given DAG in the same graphical structure as the DAG.

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.

Chapter 4

Results and Analysis

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.

4.1 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 4.1. 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

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	**0.182** ± 0.006	**0.799** ± 0.010
DNA	0.198 ± 0.005	0.758 ± 0.010

Table 4.1: Mean and standard deviation of metamodel 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.

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 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 metadata. 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.

4.2 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 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 requires that the metamodel process only the new pipeline instead of all previously considered pipelines. This advantage applies to AutoML systems like Auto-sklearn [18], TPOT [35], and AlphaD3M [15, 16], where pipelines are generated dynamically at runtime.

We measure the quality of rankings made by the metamodels using the Spearman correlation coefficient (following [5]), 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 (Section 3.2). The ranking results of our metamodels are shown in Table 4.2.

The metamodels that gave the best estimates of performance (Table 4.1) 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.

4.3 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 the top several results.

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

Table 4.2: Mean and standard deviation of metamodel 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.

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 4.3, 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.

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

Table 4.3: Mean and standard deviation of metamodel 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.

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.

4.4 Why is Random so good?

As shown in Table 4.3, the Random metamodel offers a competitive baseline in terms of choosing at least one high scoring pipeline with its low Regret@k score. It seems counter-intuitive that the most naive model outperforms the models that get to look at the training data. We do note that these models were trained to estimate pipeline performance directly by minimizing the L2 loss, not to select the best pipeline of a collection. Still, we explore why Random might be such a strong baseline.

Pipeline scores (F1 macro) lie on the interval $[0, 1]$. Suppose for some dataset that the pipeline scores were distributed uniformly across that interval, with the best pipeline having a perfect score of 1. Then choosing k pipelines at random, or in other words ranking the pipelines randomly and choosing the top k from that ranking, would likely yield k pipelines that were more or less uniformly distributed on $[0, 1]$. The best of the k would, of course, be close to 1. More formally, the score of the best pipeline under these assumptions is known as the *largest-order statistic* [11], whose expected value, when sampling from a uniform distribution, is $k/(k + 1)$.

Under these assumptions, when k is 100, we would expect the Random metamodel to pick a pipeline with a score of $100/101 \approx 0.990$. The Regret@k, assuming that a pipeline with a perfect score exists, would be 0.010. We observe in Table 4.3 that the Random metamodel performs similarly. While pipeline scores may not be uniformly distributed, this analysis suggests that sampling pipelines uniformly at random will generate at least one that is close to the best possible pipeline for a given dataset.

Chapter 5

Conclusion

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 hypothetical practical 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 [48, 58], 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.

References

[1] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 39–48, 2016.

[2] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

[3] Besim Bilalli, Alberto Abelló, and Tomas Aluja-Banet. On the predictive power of meta-features in openml. *International Journal of Applied Mathematics and Computer Science*, 27(4):697–712, 2017.

[4] Pavel Brazdil, Christophe Giraud-Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: Applications to data mining.* Springer Science & Business Media, 2008.

[5] Pavel B Brazdil, Carlos Soares, and Joaquim Pinto Da Costa. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277, 2003.

[6] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[7] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[8] Boyuan Chen, Harvey Wu, Warren Mo, Ishanu Chattopadhyay, and Hod Lipson. Autostacker: A compositional evolutionary learning system. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 402–409, 2018.

[9] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[10] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7(Mar): 551–585, 2006.

[11] Herbert A. David and Haikady N. Nagaraja. *Order Statistics.* John Wiley & Sons Inc., New York, United States, 3 edition, 2005. ISBN 9780471722168.

[12] Casey Davis and Christophe Giraud-Carrier. Annotative experts for hyperparameter selection. In *AutoML Workshop at ICML*, 2018. URL `https://sites.google.com/site/automl2018icml/accepted-papers`.

[13] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[14] Petros Drineas and Michael W Mahoney. On the nyström method for approximating a gram matrix for improved kernel-based learning. *journal of machine learning research*, 6 (Dec):2153–2175, 2005.

[15] Iddo Drori, Yamuna Krishnamurthy, Remi Rampin, Raoni Lourenço, Jorge One, Kyunghyun Cho, Claudio Silva, and Juliana Freire. Alphad3m: Machine learning pipeline synthesis. In *AutoML Workshop at ICML*, 2018.

[16] Iddo Drori, Yamuna Krishnamurthy, Raoni Lourenco, Remi Rampin, Kyunghyun Cho, Claudio Silva, and Juliana Freire. Automatic machine learning by pipeline synthesis using model-based reinforcement learning and a grammar. *arXiv preprint arXiv:1905.10345*, 2019.

[17] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

[18] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, pages 2962–2970, 2015.

[19] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Initializing bayesian hyperparameter optimization via meta-learning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[20] Matthias Feurer, Benjamin Letham, and Eytan Bakshy. Scalable meta-learning for bayesian optimization. *arXiv preprint arXiv:1802.02219*, 2018.

[21] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[22] Nicolo Fusi, Rishit Sheth, and Melih Elibol. Probabilistic matrix factorization for automated machine learning. In *Advances in neural information processing systems*, pages 3348–3357, 2018.

[23] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[24] Pieter Gijsbers, Joaquin Vanschoren, and Randal S Olson. Layered tpot: Speeding up tree-based pipeline optimization. *arXiv preprint arXiv:1801.06007*, 2018.

[25] Onur Guzey, Gihad Sohsah, and Muhammed Unal. Classification of word levels with usage frequency, expert opinions and machine learning, October 2014. URL `https://doi.org/10.5281/zenodo.12501`.

[26] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[27] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

[28] Aapo Hyvarinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE transactions on Neural Networks*, 10(3):626–634, 1999.

[29] Hadi S Jomaa, Lars Schmidt-Thieme, and Josif Grabocka. Dataset2vec: Learning dataset meta-features. *arXiv preprint arXiv:1905.11063*, 2019.

[30] Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *The Journal of Machine Learning Research*, 18(1):826–830, 2017.

[31] M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

[32] Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015.

[33] Mohamed Maher and Sherif Sakr. Smartml: A meta learning-based framework for automated selection and hyperparameter tuning for machine learning algorithms. In *EDBT: 22nd International Conference on Extending Database Technology*, Mar 2019.

[34] Mitar Milutinovic. *Towards Automatic Machine Learning Pipeline Design*. PhD thesis, UC Berkeley, 2019.

[35] Randal S Olson and Jason H Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*, pages 66–74, 2016.

[36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, and et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

[38] Fábio Pinto, Carlos Soares, and João Mendes-Moreira. Towards automatic generation of metafeatures. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 215–226. Springer, 2016.

[39] John C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *ADVANCES IN LARGE MARGIN CLASSIFIERS*, pages 61–74. MIT Press, 1999.

[40] Herilalaina Rakotoarison, Marc Schoenauer, and Michèle Sebag. Automated machine learning with monte-carlo tree search. *arXiv preprint arXiv:1906.00170*, 2019.

[41] Matthias Reif. A comprehensive dataset for evaluating approaches of various meta-learning tasks. In *ICPRAM (1)*, pages 273–276, 2012.

[42] Robert Harry Riffenburgh. *Linear discriminant analysis.* PhD thesis, Virginia Polytechnic Institute, 1957.

[43] Adriano Rivolli, Luís PF Garcia, Carlos Soares, Joaquin Vanschoren, and André CPLF de Carvalho. Characterizing classification datasets: A study of meta-features for meta-learning. *arXiv preprint arXiv:1808.10406*, 2018.

[44] Brandon Schoenfeld, Christophe Giraud-Carrier, Mason Poggemann, Jarom Christensen, and Kevin Seppi. Preprocessor selection for machine learning pipelines. *arXiv preprint arXiv:1810.09942*, 2018.

[45] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International conference on artificial neural networks*, pages 583–588. Springer, 1997.

[46] Zeyuan Shang, Emanuel Zgraggen, Benedetto Buratti, Ferdinand Kossmann, Philipp Eichmann, Yeounoh Chung, Carsten Binnig, Eli Upfal, and Tim Kraska. Democratizing data science through interactive curation of ml pipelines. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1171–1188, 2019.

[47] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[48] Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. N-ary relation extraction using graph state lstm. *arXiv preprint arXiv:1808.09101*, 2018.

[49] Philip H Swain and Hans Hauska. The decision tree classifier: Design and potential. *IEEE Transactions on Geoscience Electronics*, 15(3):142–147, 1977.

[50] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855, 2013.

[51] Joaquin Vanschoren. Meta-learning. In *Automated Machine Learning*, pages 35–61. Springer, Cham, 2019.

[52] Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[54] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Learning hyperparameter optimization initializations. In *2015 IEEE international conference on data science and advanced analytics (DSAA)*, pages 1–10. IEEE, 2015.

[55] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[56] Jun won Lee and Christophe Giraud-Carrier. Predicting algorithm accuracy with a small set of effective meta-features. In *2008 Seventh International Conference on Machine Learning and Applications*, pages 808–812. IEEE, 2008.

[57] Raymond E Wright. Logistic regression. 1995.

[58] Xiaodan Zhu, Parinaz Sobhani, and Hongyu 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*, pages 917–926, 2016.

Appendices

Appendix A

Datasets

Dataset from Zenodo [25]:

- Word Level Classification Dataset

Datasets from UCI Machine Learning Repository [13, 31]:

- Hepatitis Dataset
- Breast Cancer Wisconsin Diagnostic Dataset
- DrivFace Dataset
- Breast Cancer Wisconsin Original Dataset
- SPECT Heart Dataset

Datasets from OpenML [52]:

- Abalone Dataset (3 classes)
- ADA Agnostic Dataset
- ADA Prior Dataset
- Adult Dataset
- Allrep Dataset
- Analcatdata Authorship Dataset
- Analcatdata Birthday Dataset
- Analcatdata BroadwayMult Dataset
- Analcatdata DMFT Dataset
- Analcatdata GermanGSS Dataset
- Analcatdata GSSSexSurvey Dataset
- Analcatdata HallOfFame Dataset

- Analcatdata Lawsuit Dataset
- Analcatdata Reviewer Dataset
- Anneal Dataset
- Arsenic Female Bladder Dataset
- Arsenic Male Bladder Dataset
- Artificial Characters Dataset
- Audiology Dataset
- Australian Dataset
- AutoHorse Dataset
- AutoUniv-Au1-1000 Dataset
- AutoUniv-Au4-2500 Dataset
- AutoUniv-Au6-1000 Dataset
- AutoUniv-Au6-400 Dataset
- AutoUniv-Au6-750 Dataset
- AutoUniv-Au7-1100 Dataset
- AutoUniv-Au7-500 Dataset
- AutoUniv-Au7-700 Dataset
- Backache Dataset
- Badges2 Dataset
- Balance Scale Dataset
- Banana Dataset
- Bank Marketing Dataset
- Banknote Authentication Dataset
- Baseball Dataset
- Biomed Dataset
- Blood Transfusion Service Center Dataset
- Breast Cancer Dataset
- BreastTumor Dataset
- Breast W Dataset

- CalendarDOW Dataset
- Car Dataset
- Cardiotocography Dataset
- Cardiotocography Dataset (3 classes)
- Cars Dataset
- CastMetal1 Dataset
- Chscase Funds Dataset
- Chscase Vine2 Dataset
- Chscase Whale Dataset
- Click Prediction Small Dataset
- Climate Model Simulation Crashes Dataset
- CMC Dataset
- Colic Dataset
- Collins Dataset
- Corral Dataset
- CostaMadre1 Dataset
- Credit Approval Dataset
- Credit G Dataset
- CRX Dataset
- Cylinder Bands Dataset
- Dermatology Dataset
- Diabetes Dataset
- Diggle Table A2 Dataset
- Dresses Sales Dataset
- E. Coli Dataset
- Eucalyptus Dataset
- Eye Movements Dataset
- First Order Theorem Proving Dataset
- FishCatch Dataset

- Flags Dataset
- Flare Dataset
- GAMETES Epistasis 2 Way 20atts 0.1H EDM 1 1 Dataset
- GAMETES Epistasis 2 Way 20atts 0.4H EDM 1 1 Dataset
- GAMETES Epistasis 3 Way 20atts 0.2H EDM 1 1 Dataset
- GAMETES Heterogeneity 20atts 1600 Het 0.4 0.2 50 EDM 2 001 Dataset
- GAMETES Heterogeneity 20atts 1600 Het 0.4 0.2 75 EDM 2 001 Dataset
- Glass Dataset
- Grub Damage Dataset
- Haberman Dataset
- Hayes Roth Dataset
- Heart C Dataset
- Hepatitis Dataset
- Hill Valley Dataset
- Housing Dataset
- Indian Liver Patient Dataset
- Ionosphere Dataset
- Irish Dataset
- Japanese Vowels Dataset
- JM1 Dataset
- KC1 Dataset
- KC2 Dataset
- KC3 Dataset
- King+Rook versus King+Pawn Dataset
- Led24 Dataset
- LED Display Dataset
- Lowbwt Dataset
- Machine CPU Dataset
- MC2 Dataset

- MeanWhile1 Dataset
- MegaWatt1 Dataset
- MFeat Fourier Dataset
- MFeat Karhunen Dataset
- MFeat Morphological Dataset
- MFeat Zernike Dataset
- MiceProtein Dataset
- M of N 3 7 10 Dataset
- Monks Problems 1 Dataset
- Monks Problems 3 Dataset
- MW1 Dataset
- Nursery Dataset
- One Hundred Plants Margin Dataset
- One Hundred Plants Texture Dataset
- OptDigits Dataset
- Ozone Level 8hr Dataset
- Page Blocks Dataset
- Parity 5 Plus 5 Dataset
- Parkinsons Dataset
- PC1 Dataset
- PC3 Dataset
- PC4 Dataset
- PenDigits Dataset
- Phoneme Dataset
- PieChart1 Dataset
- Pima Dataset
- PizzaCutter1 Dataset
- Planning Relax Dataset
- PopularKids Dataset

- PRNN Crabs Dataset
- PRNN Synth Dataset
- pwlinear Dataset
- QSAR Biodegradation Dataset
- Ringnorm Dataset
- RMfTSA Cyprus Tourist Arrivals Dataset
- RMfTSA Sleep Dataset
- Robot Failures LP5 Dataset
- South Africa Heart Disease Dataset
- Seeds Dataset
- Segment Dataset
- Seismic Bumps Dataset
- Servo Dataset
- Smartphone Based Recognition Of Human Activities Dataset
- Solar Flare 1 Dataset
- Solar Flare 2 Dataset
- Sonar Dataset
- Soybean Dataset
- Spambase Dataset
- SPECTF Heart Dataset
- Splice Dataset
- Steel Plates Fault Dataset
- Synthetic Control Dataset
- Tamil Nadu Electricity Dataset
- Teaching Assistant Dataset
- Tecator Dataset
- Texture Dataset
- Thoracic Surgery Dataset
- Thyroid Allbp Dataset

- Thyroid Allhyper Dataset
- Thyroid Ann Dataset
- Thyroid Dis Dataset
- Thyroid New Dataset
- Tic Tac Toe Dataset
- Titanic Dataset
- Tokyo1 Dataset
- Triazines Dataset
- Twonorm Dataset
- User Knowledge Dataset
- Vehicle Dataset
- Vertebra Column Dataset
- Vertebra Column Dataset (2 classes)
- Volcanoes A1 Dataset
- Volcanoes A2 Dataset
- Volcanoes A3 Dataset
- Volcanoes A4 Dataset
- Volcanoes B1 Dataset
- Volcanoes B4 Dataset
- Volcanoes D1 Dataset
- Vote Dataset
- Vowel Dataset
- Wall Robot Navigation Dataset
- Wall Robot Navigation Dataset (3 attributes)
- Wall Robot Navigation Dataset (5 attributes)
- Waveform 5000 Dataset
- Breast Cancer Wisconsin Diagnostic Dataset
- Wholesale Customers Dataset
- Wilt Dataset

- Wine Quality White Dataset
- XD6 Dataset

Appendix B

Computed Metafeatures

1. Number Of Instances
2. Number Of Features
3. Number Of Numeric Features
4. Number Of Categorical Features
5. Ratio Of Numeric Features
6. Ratio Of Categorical Features
7. Number Of Classes
8. Mean Class Probability
9. Skew Class Probability
10. Kurtosis Class Probability
11. Min Class Probability
12. Quartile 1 Class Probability
13. Quartile 2 Class Probability
14. Quartile 3 Class Probability
15. Max Class Probability
16. Minority Class Size
17. Majority Class Size
18. Dimensionality
19. Number Of Missing Values
20. Ratio Of Missing Values
21. Class Entropy
22. Naive Bayes Err Rate

23. Naive Bayes Kappa
24. kNN 1N Err Rate
25. kNN 1N Kappa
26. Decision Stump Err Rate
27. Decision Stump Kappa
28. Random Tree Depth 1 Err Rate
29. Random Tree Depth 1 Kappa
30. Random Tree Depth 2 Err Rate
31. Random Tree Depth 2 Kappa
32. Random Tree Depth 3 Err Rate
33. Random Tree Depth 3 Kappa
34. Mean Decision Tree Level Size
35. Total Metafeature Compute Time
36. Skew Decision Tree Level Size
37. Kurtosis Decision Tree Level Size
38. Min Decision Tree Level Size
39. Decision Tree Node Count
40. Max Decision Tree Attribute
41. Max Decision Tree Level Size
42. Linear Discriminant Analysis Kappa
43. Quartile 1 Decision Tree Level Size
44. Quartile 2 Decision Tree Level Size
45. Quartile 3 Decision Tree Level Size
46. Mean Decision Tree Branch Length
47. Skew Decision Tree Branch Length
48. Decision Tree Leaf Count
49. Decision Tree Height
50. Min Decision Tree Branch Length
51. Max Decision Tree Branch Length

52. Decision Tree Width
53. Mean Decision Tree Attribute
54. Kurtosis Decision Tree Branch Length
55. Number Of Features With Missing Values
56. Ratio Of Features With Missing Values
57. Number Of Instances With Missing Values
58. Ratio Of Instances With Missing Values
59. Standard Deviation Decision Tree Attribute
60. Skew Decision Tree Attribute
61. Kurtosis Decision Tree Attribute
62. Min Decision Tree Attribute
63. Quartile 1 Decision Tree Attribute
64. Quartile 2 Decision Tree Attribute
65. Quartile 3 Decision Tree Attribute
66. Standard Deviation Class Probability
67. Linear Discriminant Analysis Err Rate
68. Quartile 1 Decision Tree Branch Length
69. Quartile 2 Decision Tree Branch Length
70. Quartile 3 Decision Tree Branch Length
71. Standard Deviation Decision Tree Level Size
72. Standard Deviation Decision Tree Branch Length