Brigham Young University
BYU ScholarsArchive

All Theses and Dissertations

2017-06-01

Informing the use of Hyper-Parameter Optimization Through Meta-Learning

Samantha Corinne Sanders
Brigham Young University

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Computer Sciences Commons

BYU ScholarsArchive Citation

Sanders, Samantha Corinne, "Informing the use of Hyper-Parameter Optimization Through Meta-Learning" (2017). *All Theses and Dissertations*. 6392.
https://scholarsarchive.byu.edu/etd/6392

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.

Informing the Use of Hyper-Parameter Optimization Through Meta-Learning

Samantha Corinne Sanders

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

Master of Science

Christophe Giraud-Carrier, Chair
Tony Martinez
Bryan Morse

Department of Computer Science

Brigham Young University

Copyright © 2017 Samantha Corinne Sanders

All Rights Reserved

ABSTRACT

Informing the Use of Hyper-Parameter Optimization Through Meta-Learning

Samantha Corinne Sanders
Department of Computer Science, BYU
Master of Science

One of the challenges of data mining is finding hyper-parameters for a learning algorithm that will produce the best model for a given dataset. Hyper-parameter optimization automates this process, but it can still take significant time. It has been found that hyper-parameter optimization does not always result in induced models with significant improvement over default hyper-parameters, yet no systematic analysis of the role of hyper-parameter optimization in machine learning has been conducted. We propose the use of meta-learning to inform the decision to optimize hyper-parameters based on whether default hyper-parameter performance can be surpassed in a given amount of time. We will build a base of meta-knowledge, through a series of experiments, to build predictive models that will assist in the decision process.

Keywords: Meta-learning, Hyper-parameter optimization

ACKNOWLEDGMENTS

Thanks to my family for supporting me, Christophe for being a great advisor, and BYU's supercomputer for doing all of the grunt work.

Table of Contents

List of Figures **vi**

List of Tables **vii**

1 Introduction **1**

1.1 Background 1

1.2 Significance 2

1.2.1 Researchers 2

1.2.2 Practitioners 2

1.2.3 Time Limitations 2

1.2.4 Big Data 3

1.3 Thesis Statement 3

2 Related Work **4**

2.1 Meta-learning 4

2.2 AutoML 4

2.3 Bias-Variance Trade-off 5

2.4 Impact of Hyper-parameter Optimization 5

3 Experimental Setup **8**

3.1 Experiment Components 8

3.1.1 Optimization Method 8

3.1.2 Data 10

3.1.3 Algorithms 10
3.1.4 Performance Measure 10
3.2 Analysis on a per-algorithm basis 12
3.2.1 Method 13
3.3 Optimization on a budget 14
3.3.1 Method 14

4 Results **15**
4.1 Analysis on a per-algorithm basis 15
4.2 Optimization on a budget 18

5 Conclusion **25**

References **27**

Appendices **29**
A Datasets 29
B Meta-features 31

List of Figures

4.1 Average percent improvement over default hyper-parameters 16

4.2 Average percent improvement over default hyper-parameters, zoomed in. . . 16

4.3 Average gap between default hyper-parameters confidence interval and optimized hyper-parameters confidence interval. 17

4.4 Meta-learner performance results for predicting MAUC improvement after hyper-parameter optimization. 23

4.5 Time to meet default upper bound 24

List of Tables

3.1 Decision Tree Hyper-parameters 11

3.2 SVM Hyper-parameters 11

3.3 MLP Hyper-parameters 12

4.1 Meta-learner performance results for predicting MAUC improvement after hyper-parameter optimization. 17

4.2 Average runtimes for meeting the default upper bound in seconds. 18

4.3 Linear regression performance results for predicting runtimes. 18

4.4 Top five meta-features for predicting MLP runtimes. 19

4.5 Top five meta-features for predicting SVM runtimes. 20

4.6 MLP runtime classifier results 20

4.7 SVM runtime classifier results 21

4.8 Decision Tree runtime classifier results 21

Chapter 1

Introduction

1.1 Background

With growing interest in data science and the proliferation of algorithms, practitioners are faced with the challenge of deciding what algorithm works well where. Currently there is no hard and fast method for choosing the best algorithm for a given dataset, so practitioners are left to fumble around a dark room searching for the best algorithm. To help, some researchers have turned to meta-learning, the use of machine learning to build algorithm selection models from data about the application of machine learning [3]. Since most algorithms also have hyper-parameters that can be adjusted, the selection of adequate hyper-parameter values adds yet another dimension to this search problem. Work on meta-learning for algorithm selection has, however, often been criticized because it mostly considers only the default hyper-parameter settings of the base learning algorithms.

The assumption behind this criticism is that hyper-parameter settings could have a significant impact on the generalization accuracy of learned classifiers, and by ignoring hyper-parameter settings practitioners are getting suboptimal results. Surprisingly, little has been done to validate this claim, which has at once been used to discredit past work and to justify large research efforts in hyper-parameter optimization. What if this hyper-parameter optimization claim does not hold in general? We could be turning down valid research that uses default hyper-parameters. We could also be wasting considerable time and computational effort finding optimal hyper-parameters for datasets and/or algorithms that are not sensitive

to hyper-parameter settings. In reality, we do not know at what cost, by how much, or for what kinds of datasets and algorithms hyper-parameter optimization makes a difference.

1.2 Significance

1.2.1 Researchers

Information about the potential impact of hyper-parameter optimization is useful for researchers and reviewers. When designing experiments, researchers would like to make informed decisions about whether to use hyper-parameter optimization. When reviewing the work of others, researchers must be able to offer accurate and well-founded critiques. If we know when hyper-parameter optimization makes a difference, then we stand on firm ground in a decision to accept or reject a piece of research based on whether hyper-parameters are optimized or not.

1.2.2 Practitioners

The data deluge has only begun and the demand for data mining is increasing, so another group of people affected by this research are non-expert practitioners. By building automatic advice strategies, we lower the barrier of entry into data mining. As a result, those for whom the technology is intended may actually have access to it in an economical and direct way, without having to resort to expensive consultants.

1.2.3 Time Limitations

This research has particular impact in areas where time is a limited resource and data analysis is critical for best performance. Consider the business world where quick results are important for competitiveness. If practitioners know ahead of time that optimizing hyper-parameters is not going to make a difference, then they could potentially save hours or days, depending on the dataset, and make quicker decisions.

1.2.4 Big Data

Consider now a practitioner dealing with a large dataset, one that would be regarded as "Big Data". How important would it be to know whether hyper-parameter settings are significant? Users could save massive amounts of time if they knew that their dataset was not sensitive to hyper-parameter settings for certain learning algorithms. Even if it was common practice to perform hyper-parameter optimization with small datasets, datasets that fit in memory, we may have to rethink the possibility of this practice in the realm of Big Data. Often, commonly used practices and methods that we use on small datasets cannot be easily applied to distributed data. Such is the case with the practice of performing hyper-parameter optimization. If hyper-parameter optimization takes a long time with current methods and small datasets, it will take significantly longer with huge datasets. This research is the beginning of thinking more intelligently about when we perform hyper-parameter optimization.

1.3 Thesis Statement

When it comes to hyper-parameter optimization, not all datasets are created equal nor are all classification learning algorithms, and thus we can use meta-learning to predict when, to what extent, and at what cost, hyper-parameter optimization for a given learning algorithm/dataset combination will have an impact on the generalization performance of the resulting model.

Chapter 2

Related Work

2.1 Meta-learning

In meta-learning we are generally concerned with matching machine learning algorithms and datasets [3]. This is done using metaknowledge, which is knowledge about the learning process. Such information includes statistics about the dataset and how different kinds of datasets respond to different algorithms. We also use meta-learning here, but with the goal of matching datasets with the decision to optimize or not, rather than matching datasets with algorithms. The metaknowledge-base for this meta-learning problem is quite limited and we must expand it.

2.2 AutoML

The question of how to do hyper-parameter optimization in the general case is being actively explored (e.g., see [2, 7, 12, 13]). There are a number of optimization approaches, but none of them take the initial step to determine if parameter optimization will improve performance. AutoML [5] recently emerged as an area of research with the goal to make machine learning more accessible to non-experts. The goal is to automate the machine learning process including preprocessing the data, selecting appropriate features, selecting a model, optimizing hyper-parameters, and analyzing the final results. Some examples of AutoML implementations include AutoWEKA [15] and HPOlib [5]. Our research takes advantage of meta-learning techniques to build predictive models that will help with the prerequisite decision of whether to optimize parameters.

2.3 Bias-Variance Trade-off

Machine learning algorithms all have a bias: an assumption they make about the data. For example, linear regression models have the bias that all data can be modeled by a hyperplane, and multilayer perceptrons have the bias that all data can be modeled by non-linear functions. By the No Free Lunch theorems [17], however, there is no one bias, or learning algorithm, that works best in every situation. Thus we must find the bias that works best for each situation. Each bias is accompanied by a certain amount of variance, or precision in the induced models. We often talk about the bias error, or the amount of error in the model resulting from the assumptions of the learning algorithm [4]. Linear regression models have a large bias error and less variance error in the results of the model. Most of the error comes from the learning algorithms bias that the data can be fit with a line which may not be a good representation of the data. The error due to variance is low because the induced models do not change significantly when a different training set from the same population is used. Multilayer perceptrons have a smaller bias error because the assumptions made by the learning algorithm are more "flexible, but more variance error in the induced model. Changing the training set for a multilayer perceptron has a much more significant impact on the induced model than it would for the linear regression model. We would expect that models with smaller bias error and larger variance error, such as the multilayer perceptron, would benefit more from hyper-parameter optimization than models with larger bias error and small variance error, such as a linear regression model, because the induced models change more drastically with smaller changes.

2.4 Impact of Hyper-parameter Optimization

In [14], the authors address explicitly for the first time the issue of hyper-parameter optimization. They consider 466 datasets and for each, compute the difference in AUC among 20 algorithms between their default hyper-parameter setting and the best possible hyper-

parameter setting after optimization. While they seem to focus on non-zero difference in the aggregate, their results suggest that the impact of hyper-parameter optimization is highly variable across datasets. For 19% of the datasets there was no improvement over default hyper-parameters and for 95% of the datasets there was less than 5% improvement over default hyper-parameters for all algorithms. This work is a good starting point because it shows that hyper-parameter optimization does not have a universal effect on datasets, but it does not uncover any information that would be useful for making the decision whether to hyper-perform parameter optimization.

The question of when to perform hyper-parameter optimization is glossed over in this study. They simply saw that in most cases there was some performance improvement and decided that hyper-parameter optimization is beneficial in every case. Not every dataset saw improvement from hyper-parameter optimization, and some datasets had very little improvement, so we would like to know when we can expect improvement for a particular dataset and how much. Such information could significantly decrease the amount of time spent on possibly unnecessary hyper-parameter optimization.

Information that is not provided in [14] is how individual algorithms respond to hyper-parameter optimization. The differences in performance observed in [14] are computed from among the best in 20 algorithms, which means that the best optimized version could be obtained with one algorithm, while the best default version for the same dataset could be obtained with another. It is likely, however, that some classification learning algorithms are more sensitive to hyper-parameter settings. Understanding how sensitive individual learning algorithms are to hyper-parameter optimization could influence the way that we approach hyper-parameter optimization. For example, if we knew that SVM had little sensitivity to hyper-parameter settings then we could decrease the search space, and thus search time. We would omit SVM from the search and just compare the output model from hyper-parameter optimization with the default settings for SVM. Such a method could be used when only close-to-optimal results are needed and time is a major consideration.

This study also does not consider the dimension of time on a per-algorithm basis. We would like to know how long it takes for individual algorithms to reach a certain level of performance. Indeed, optimization is an expensive process, and one may be interested in knowing how much improvement (if any) may be expected within a given time budget, or how much time should be invested to reach some expected level of improvement. Understanding when and how much improvement we expect from hyper-parameter optimization, how sensitive individual algorithms are to hyper-parameter optimization, and how long it takes for those algorithms to reach a certain level of performance will give us a basis for how and when to perform hyper-parameter optimization. This knowledge will enrich our understanding of the behavior of learning algorithms and inform our hyper-parameter optimization decisions.

Another recently published paper [11] begins to analyze the hyper-parameter optimization results obtained in [14]. They used meta-learning techniques to build models that can predict when a dataset will have performance improvement over some threshold using the same 466 datasets as [14]. They generated meta-features for each of the datasets and labeled each dataset as 0 if it was below the performance threshold and 1 if it was above the performance threshold. Then they used machine learning methods for inducing a model with the meta-data. With thresholds of 1.5% and 2.5% improvement their model could claim 83.21% and 73.60%, respectively, of the performance improvement that would be obtained by performing hyper-parameter optimization for all of the datasets. Not only did this study begin to build models for predicting when hyper-parameter optimization is expected to improve performance by a certain amount, but it also makes some conclusions about which meta-features are informative to predict whether a dataset will have performance improvement. However, there again, no per-algorithm analysis is offered, nor is there a discussion of budget.

Other than these two studies, we have not found other attempts at addressing the validity of the parameter optimization claim.

Chapter 3

Experimental Setup

As stated above, our goal is to help researchers and practitioners make decisions about when and how to perform parameter optimization. To accomplish this goal this project has two main parts: (1) analysis on a per-algorithm basis and (2) optimization on a budget.

3.1 Experiment Components

3.1.1 Optimization Method

Popular methods of optimization include grid search, random search [1], particle swarm optimization (PSO), Bayesian optimization [10], and genetic algorithms. In [14] the authors use PSO. We have decided not to use PSO because it is best suited for continuous parameters, and not all of the parameters we want to optimize are continuous. Actually, there are often combinations of continuous and discrete parameters for each algorithm and it seems more reasonable to seek an optimization method that can deal with that.

We initially tried a Bayesian optimization approach using a software package called Auto-WEKA [15] as it handles both continuous and discrete parameters. After extensive experimentation with of the package, however, we found that it did not meet all of the needs of this project. We used a similar approach to the setup in [8] so we used a simple genetic algorithm for hyper-parameter optimization as this is a common approach for doing a pseud-random search. We chose the hyper-parameters for the genetic algorithm based on other implementations of genetic algorithms and no tuning was performed with the genetic

algorithm hyper-parameters for this experiment. The following is the setup we used for the genetic algorithm:

Population: The initial population of 100 individuals contained an individual with default hyper-parameters. This guarantees that the end solution would be at least as good as default hyper-parameters.

Selection: Tournament selection with 5 individuals per selection. We also employed elitism, carrying over the best individual from the previous generation into the next.

Crossover: Uniform crossover rate of 0.5

Mutation: 0.015 chance of mutating each gene

- Floating-point hyper-parameter mutation: We sample from a Gaussian distribution with mean set to the current value of the gene and the standard deviation is specified at the creation of the floating point gene. Sampling is repeated until a value within the specified range for the hyper-parameter is selected.
- Integer hyper-parameter mutation: As with floating-point parameters we sample from a Gaussian distribution with mean set to the current value of the gene and the standard deviation is specified at the creation of the integer gene. The value sampled is rounded to the nearest integer. Sampling is repeated until a value within the specified range for the hyper-parameter is selected.
- List hyper-parameter mutation: These gene types are used for hyper-parameters that have a mixture of types, ex: [None, 10, 100, 1000] or hyper-parameters that have string values. To mutate, we randomly select a member of the list of hyper-parameter settings that is different from the current setting.

Fitness Function: We use 10-fold cross-validation for each algorithm with Multi-class AUC (MAUC) as the result metric.

3.1.2 Data

We considered using the same 466 datasets used in [14], but after looking closer at those datasets we felt it better to build our own base of datasets to test with. All of the 466 datasets were binary classification problems, even if the dataset was not naturally a binary classification problem (only instances from the top two classes were used). We felt that it was important for us to do as little pre-processing as necessary in order to account for a wide variety of dataset types. We gathered all of our 229 data sets from OpenML [16] with the requirement that each dataset has at least 100 instances. See appendix for a list of datasets used.

3.1.3 Algorithms

We consider three algorithms in this study: SVM, MLP, and Decision Tree (an optimized version of CART). We chose these algorithms because they are widely used, they come from distinct classes of learning, and they have a significant number of parameters. We use the algorithms as found in the scikit-learn library [9] since scikit-learn is becoming an increasingly popular machine learning library, and we wanted our results to be relevant to the practitioner as well as to the researcher. We used as many hyper-parameters for each algorithm as were provided by the sci-kit learn package and that made sense for our experiment. The hyper-parameter value ranges were chosen to be within a common range of use and then a little beyond that if the hyper-parameter type allowed. See Table 3.1, Table 3.2, and Table 3.3 for a complete listing of the hyper-parameters used.

3.1.4 Performance Measure

In [14] they binarized all of their datasets, allowing them to be able to use AUC as their performance measure. They note in [14] that they used AUC because it is less sensitive to data skew, which was present in some data sets. We also wanted to use a performance measure that is less sensitive to data skew, but because we did not want to modify our

Parameter Name	Data Type	Default Value	Range
Criterion	List	gini	[gini, entropy]
Splitter	List	best	[best, random]
Max. Features	List	None	[None, sqrt, log2]
Max. Depth	List	None	[None, 10, 100, 1000]
Min. Samples Split	Integer	2	[2 - 10]
Min. Samples Leaf	Integer	1	[1 - 20]
Min. Weight Fraction Leaf	Float	0.0	[0.0 - 0.5] std: 0.25
Max. Leaf Nodes	List	None	[None, 10, 100, 1000]

Table 3.1: Decision Tree Hyper-parameters

Parameter Name	Data Type	Default Value	Range
C	Float	1.0	[0.0 - 1.0] std: 0.1
Kernel	String	rbf	[linear, poly, rbf, sigmoid]
Degree	Integer	3	[2 - 5]
Gamma	Float	0.2	[0.0 - 1.0] std: 0.1
Coef0	Float	0.0	[0.0 - 1.0] std: 0.1

Table 3.2: SVM Hyper-parameters

datasets we were not able to use the AUC metric. As an alternative, we used MAUC [6], which is the generalized version of the AUC metric which can be used for data sets with multiple (more than two) class labels.

We encountered an unforeseen limitation of MAUC metric when there were scenarios when a learning algorithm predicted all of the instances in a particular fold to be the same label, which resulted in a zero division error. We discussed this issue with the authors of [6] and determined that there was not a clear solution to this problem that would not introduce any bias. In order to move forward with our experiment we assigned individuals, or hyper-parameter settings, in the genetic algorithm with these issues a fitness value of 0 out of a maximum score of 1.0. The effect was that these hyper-parameter settings that produced the zero division error quickly dropped out of the population.

Parameter Name	Data Type	Default Value	Range
Hidden Layer Sizes	Tuple	(100,)	([1-200], [0-200], [0-200])
Activation	List	relu	[identity, logistic, tanh, relu]
Solver	List	adam	[lbgfs, sgd, adam]
Alpha	Float	1e-4	[1e-4 -1]
Batch Size	List	auto	[auto, 10, 100, 1000]
Learning Rate	List	constant	[constant, invscaling, adaptive]
Max. Iter.	Integer	200	[50, 100, 200, 500, 1000, 5000]
Tol	Float	1e-4	[1e-6 - 1e-1]
Learning Rate Init.	Float	1e-3	[1e-4 - 1]
Power T	Float	0.5	[0.01 - 1]
Warm Start	Boolean	False	[True, False]
Momentum	Float	0.9	[0 - 1]
Nesterovs Momentum	Boolean	True	[True, False]
Early Stopping	Boolean	False	[True, False]
Validation Fraction	Float	0.1	[0 - 1]
Beta 1	Float	0.9	[0, 1)
Beta 2	Float	0.999	[0, 1)
Epsilon	Float	1e-8	[1e-10 - 1e-2]

Table 3.3: MLP Hyper-parameters

3.2 Analysis on a per-algorithm basis

We will similarly re-run the analysis in [14] on a per-algorithm basis. The analysis in [14] compared the best AUC scores from 20 optimized algorithms for each dataset to the best AUC scores from the same 20 algorithms with default hyper-parameters. For each algorithm and dataset combination we want to compare the MAUC scores of the optimized algorithm to the MAUC scores of the algorithm with scikit-learn default hyper-parameters. This analysis will help us understand how individual learning algorithms respond to hyper-parameter optimization.

We suspect that some algorithms will have a greater response to hyper-parameter settings than other algorithms. We would expect, for example, that decision trees may not be as sensitive to their hyper-parameter settings because they have fewer (discrete-valued) hyper-parameters. On the other hand, we would expect Support Vector Machines, which have more (real-valued) hyper-parameters, to be more responsive to hyper-parameter settings

and thus benefit more from hyper-parameter optimization. In the case that not all algorithms respond equally to hyper-parameter optimization, we could make decisions about which algorithms make sense to optimize and when. This would reduce the search space and thus decrease the time needed for hyper-parameter optimization.

3.2.1 Method

We ran the genetic algorithm 30 times for each algorithm / data set combination each time with a random start (30 starts $\times$ 3 algorithms $\times$ 229 datasets = 20,610 runs). These experiments were performed on BYUs supercomputer allowing us to run thousands of experiments at once. The stopping condition for the genetic algorithm is either a solution is found that yields a perfect MAUC, or 24 hours has elapsed. For each run we record the best solution so far with the time it was found (relative to the time the genetic algorithm started running), the generation, and the algorithm hyper-parameters that produced that solution. We also ran the default hyper-parameters 30 times for each algorithm / data set combination to use as a baseline for the amount of improvement achieved through hyper-parameter optimization. We then calculated confidence intervals (0.95) for the end optimized results with the 30 runs for each data set / algorithm combination for both optimized and default results.

If we were to find no overlap between the default confidence interval and the optimized confidence interval, we would label that data set as "optimize meaning that given time, optimization would yield improvement over default hyper-parameters. Otherwise, that data set would be labeled "dont optimize meaning that there may not be improvement over default hyper-parameters after doing hyper-parameter optimization. Then after calculating meta-features for each data set we could use meta-learning to try to predict when a data set / algorithm combination would benefit from hyper-parameter optimization.

3.3 Optimization on a budget

We will further extend the analysis with a notion of budget. Indeed, optimization is an expensive process, and one may be interested in knowing how much improvement (if any) may be expected within a given time budget, or how much time should be invested to reach some expected level of improvement. Hence, performance differences will be computed along three dimensions: algorithm, dataset and time, i.e., for each algorithm, and for each dataset, we will consider the increase in MAUC as a function of time spent optimizing hyper-parameters. Such detailed information will greatly enrich our understanding of the impact of hyper-parameter optimization and will help inform practitioners decisions. If no improvement is to be expected from hyper-parameter optimization, then one would gladly save the extra computational time required to effect it.

3.3.1 Method

In our analysis on a per-algorithm basis, we were just interested in the MAUC by the time the stopping criteria was met. Here we are interested in the time at which improvement over default hyper-parameters is expected to be met. No additional data was needed here since we recorded improvement over time as the genetic algorithm ran. We then went back and gathered the 30 times at which the optimized results surpassed or equalled the upper bound of the default confidence interval for each algorithm / data set combination. With those 30 times, we calculated confidence intervals (0.95) and the average time needed to surpass the upper bound of the default confidence interval. If a particular run (one of the 30) did not surpass the upper bound of the default confidence interval, then the maximum run-time was used.

Chapter 4

Results

4.1 Analysis on a per-algorithm basis

We expected to see a number of cases in which there would be little to no improvement over default hyper-parameters as a result of hyper-parameter optimization, similar to the results found in [14]. However, we found that across the three algorithms, there were only a handful of data sets where there was no statistical improvement over default hyper-parameters, and for those data sets it was because the default hyper-parameters yielded a perfect MAUC score. So not surprisingly, if the default hyper-parameters give perfect results then do not spend time optimizing hyper-parameters, otherwise, statistically significant improvements over default hyper-parameters as a result of hyper-parameter optimization is likely. In some cases, there was significant improvement in MAUC after hyper-parameter optimization. This is especially true for MLP and SVM. In one extreme case for SVM there was a 2034% improvement over default hyper-parameters. The histograms in Figures 4.1 and 4.2 show the distribution of percent improvement over hyper-default parameters.

For each of the algorithms, there were two data sets that did not benefit at all from hyper-parameter optimization. It was not the same two data sets for all three algorithms. In each of those cases, there was no improvement from hyper-parameter optimization because the default hyper-parameters performed perfectly. However, there were seven data sets with MLP in which the default results confidence interval and the optimized results confidence interval overlapped. In other words there were seven data sets out of the 186 successfully optimized with MLP in which it would be possible that default hyper-parameters could

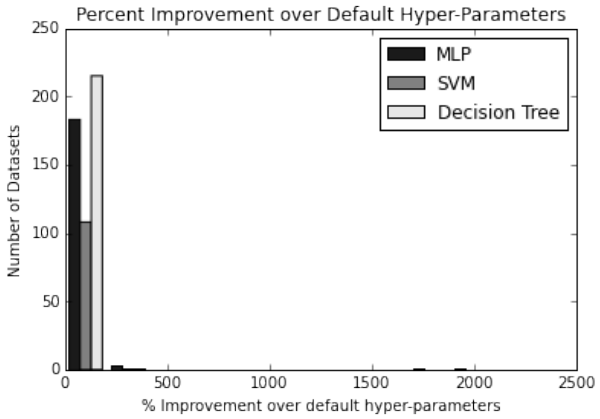


Figure 4.1: Average percent improvement over default hyper-parameters

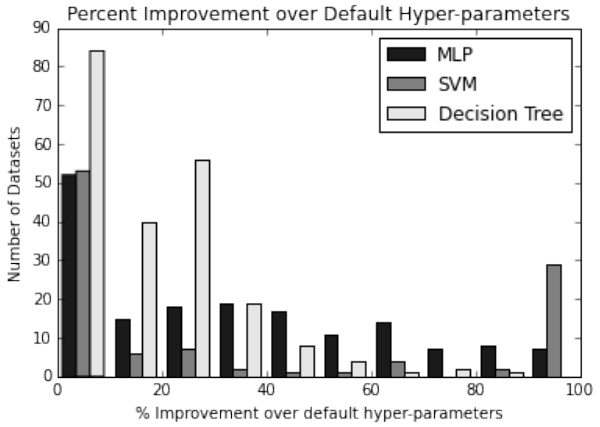


Figure 4.2: Average percent improvement over default hyper-parameters, zoomed in.

perform as well as optimized hyper-parameters. For SVM and Decision Tree, there was always a gap between the confidence intervals for default results and optimized results, meaning that there was always improvement over default hyper-parameters. See Figure 4.3.

Since it is clear that hyper-parameter optimization will almost always yield statistically significant improvements over default hyper-parameters, we went a step further and built meta-learners to predict how much improvement can be expected. We used an MLP as the meta-learner and the meta-features as listed in the appendix. We preprocessed the features with PCA, which yielded better results than using the original features. Looking at the results in Table 4.1, it would seem that the results for SVM was just about on par with MLP and Decision Tree results, but looking at Figure 4.4, it would seem that overall the predictions

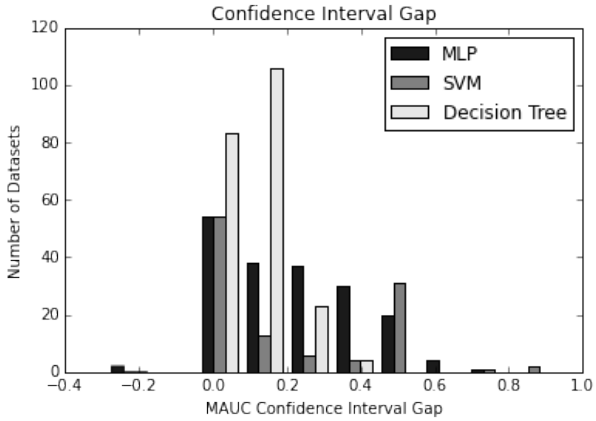


Figure 4.3: Average gap between default hyper-parameters confidence interval and optimized hyper-parameters confidence interval.

for SVM were not as close to the actual improvement values. This could be because there are not as many data points for the SVM data set so the predictions look more scattered.

Learning Algorithm	Correlation Coefficient	Mean Absolute Error	Root Mean Squared Error
MLP	0.47	0.15	0.23
SVM	0.54	0.17	0.26
Decision Tree	0.56	0.06	0.11

Table 4.1: Meta-learner performance results for predicting MAUC improvement after hyper-parameter optimization.

These meta-learners used here are built specifically for the genetic algorithm and experiment conditions used in this paper so while these particular meta-learners may not be directly usable with other optimization methods, they do show that it is possible to predict with some certainty how much improvement can be expected from hyper-parameter optimization.

On average, SVM and MLP benefited the most from hyper-parameter optimization. SVM and MLP both had an average MAUC gap between default and optimized confidence intervals of 0.21, and data sets optimized with Decision Tree had an average gap of 0.12. Some data sets, however, were not able to complete even one generation of optimization in the 24 hour time period with the genetic algorithms hyper-parameter settings. We suspect

that this is a result of the learning algorithm taking a long time to run with particular data sets due to their size. This was especially true for SVM where only 111 of the 229 datasets were able to successfully yield 30 runs that completed at least one generation.

4.2 Optimization on a budget

As expected, we found that the Decision Tree algorithm reached or surpassed the expected default results early on in the optimization process and on average, it took longer for SVM and MLP to reach the same benchmark. SVM had a lower average time to meet the default hyper-parameter benchmark than MLP, but it also had a wider range of times needed to meet the default hyper-parameter benchmark. See Table 4.2 and Figure 4.5.

Data Set	Avg. Time	Min. Avg. Time	Max.Avg. Time
MLP	11,353	39.9	58,156
SVM	9,594	1.5	64,943
Decision Tree	190	0.4	4,198

Table 4.2: Average runtimes for meeting the default upper bound in seconds.

We also did some meta-learning with a linear regression model to predict the average runtime to meet the upper bound of the default confidence interval for each learning algorithm (Decision Tree, SVM, and MLP). We used 68 meta-features, which are listed in the appendix. The purpose of using a linear regression is to find the most significant meta-features for predicting the runtimes for surpassing default hyper-parameter results. Table 4.3 summarizes the results from the learned model for each data set.

Dataset	Correlation Coefficient	Mean Absolute Error	Root Mean Squared Error
MLP	0.89	4,378	7,874
SVM	0.53	13,496	26,974
Decision Tree	-0.01	88,736,389	1,213,424,801

Table 4.3: Linear regression performance results for predicting runtimes.

The Linear Regression model had poor results with the Decision Tree data set. We tried a few other regression models with that data set but all yielded unsatisfactory results. It is important to note, however that of the 187 data sets used to generate the Decision Tree data set, 181 of them took less than 35 minutes to reach a performance level that exceeded the upper bound of the default hyper-parameter confidence interval. This means that a model for predicting runtime to reach the upper bound of the default confidence interval may not be useful in practice if it is a small time investment to optimize Decision Tree hyper-parameters anyways.

The model yielded by the SVM data set had a decent correlation coefficient, however the RMSE and MAE are not as good since the mean runtime to meet the upper bound of the default hyper-parameters confidence interval was 9,594 seconds. The results for the MLP data set were the most promising with a correlation coefficient of 0.89, indicating that there is a high positive correlation between the estimated amount of time needed to exceed the upper bound of the default hyper-parameters confidence interval and the actual time needed. The RMSE and MAE results are slightly more respectable than SVMs considering that the average runtime to exceed default hyper-parameter results was 11,353 seconds.

The tables below have the top five most significant meta-features for predicting average runtime to meet the upper bound of the default confidence interval. The table for Decision Tree is not included since the model is not useful for prediction. See Tables 4.4 and 4.5 for the top five meta-features for predicting MLP and SVM runtimes.

Rank	Sign	Meta-feature
1	+	Number of attributes
2	-	Number of numeric attributes
3	-	Standard deviation of the number of nominal attribute values
4	+	Maximum number of nominal attribute values
5	-	Maximum Decision Tree branch length

Table 4.4: Top five meta-features for predicting MLP runtimes.

Rank	Sign	Meta-feature
1	+	Number of numeric attributes
2	-	Number of attributes
3	+	Average number of nodes per Decision Tree level
4	+	Maximum Decision Tree branch length
5	+	Number of samples

Table 4.5: Top five meta-features for predicting SVM runtimes.

The results for the SVM and MLP models have three of the same top five meta-features: number of attributes, number of numeric attributes, and branch maximum. For MLP the time it takes to surpass default hyper-parameter performance is positively correlated with the number of attributes. The more attributes there are in a problem, the more weights there are to learn–which results in longer runtimes. However, the number of numeric attributes is negatively correlated with the time to reach default hyper-parameter performance. Because MLPs naturally deal with numeric attributes it could make sense that the number of numeric attributes is negatively correlated with runtime. The meaning behind the coefficients for SVM are a little less intuitive.

We also tried building a classifier to predict whether default hyper-parameter performance could be surpassed within a certain amount of time. For example, if the instance (data set) exceeded default hyper-parameter performance in less than an hour it received a label of 1 otherwise it received a label of 0. We used a Decision Tree as the meta-learner, because of its interpretable nature, to predict these values. We used 10-fold cross-validation to test the models. See Tables 4.6, 4.7, and 4.8 for the meta-learner performance results.

Runtime Cutoff	Baseline Acc. (prediction)	Accuracy	Precision	Recall	ROC Area
30 minutes	56% ($\geq$ 30 min.)	90%	0.91	0.87	0.91
1 hour	57% ($<$ 1 hour)	90%	0.91	0.92	0.91
3 hours	68% ($<$ 3 hours)	88%	0.93	0.90	0.90

Table 4.6: MLP runtime classifier results

Runtime Cutoff	Baseline Acc. (prediction)	Accuracy	Precision	Recall	ROC Area
30 minutes	58% ($\geq$ 30 min.)	88%	0.86	0.84	0.86
1 hour	50%	81%	0.82	0.80	0.78
3 hours	77% ($<$ 3 hours)	73%	0.81	0.85	0.58

Table 4.7: SVM runtime classifier results

Runtime Cutoff	Baseline Acc. (prediction)	Accuracy	Precision	Recall	ROC Area
10 seconds	52% ($\geq$ 10 sec.)	89%	0.90	0.87	0.91
60 seconds	68% ($<$ 60 sec.)	91%	0.94	0.92	0.88
30 minutes	97% ($<$ 30 min.)	97%	0.98	0.98	0.68

Table 4.8: Decision Tree runtime classifier results

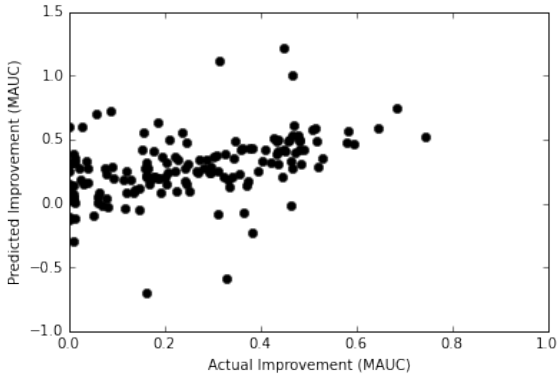
The results from the classifiers are a little more promising than the regression results. The Decision Trees for the MLP dataset did an excellent job of improving over baseline accuracy, ranging from a 20-34 percent improvement. The models for the MLP dataset also had very respectable precision and recall values. A good precision value means that nearly all of the instances labeled as needing less than a given amount of time to reach the default hyper-parameter benchmark, actually needed less than that amount of time to reach that benchmark. A good recall value is achieved when a large portion of the instances needing less than a given amount of time to reach the default hyper-parameter benchmark were actually labeled as such.

The root node for all three decision trees for the MLP data set were nn_time, which is the amount of time it takes to run the k-nearest neighbors algorithm (k = 1) for a dataset. This would indicate that nn_time is a good discriminator for the amount of time it will take to reach default hyper-parameter performance with MLP for a given dataset.

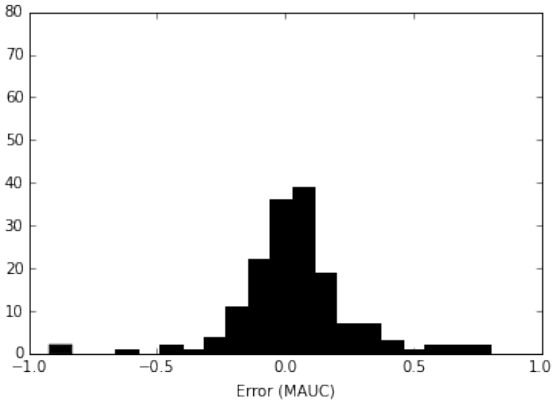
Also of note is that the Decision Tree dataset results were much better with a classifier than with a regressor. However, 97% of the datasets reached default hyper-parameter performance in under 30 minutes, so in practice these models may not be useful because it takes relatively little time to surpass default hyper-parameter performance.

The success of these models indicates that if a practitioner has a given amount of time to perform hyper-parameter optimization, they can refer to these classifiers to determine if it is likely whether they can improve over default hyper-parameters in the given amount of time that they have. This could be a great time-saver for a practitioner that may need quick results especially when trying to optimize the performance of an MLP or SVM model.

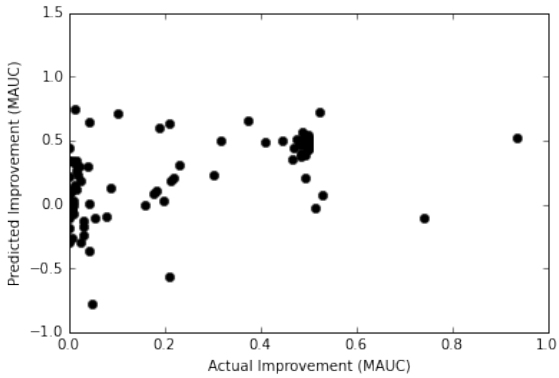
We were also interested in the correlation between the average time needed to meet the default upper bound performance and the percent improvement for each of the algorithms. There was almost zero correlation between the two values for all three algorithms: MLP had a Pearson correlation coefficient of -0.08, SVM had a Pearson correlation coefficient of -0.02, and Decision Tree had a correlation coefficient of -0.06. Therefore, the length of time it takes to surpass default hyper-parameter performance is not an indicator of the percent improvement expected. This could be due to the fact that some problems are inherently more difficult than others so even after spending significant amounts of time searching for better hyper-parameters, improvements are still small. On the other hand, it could be that the models yielded by different hyper-parameters could widely differ for some data sets. Searching over a wide range of models could lead to significant improvements over a small amount of time.



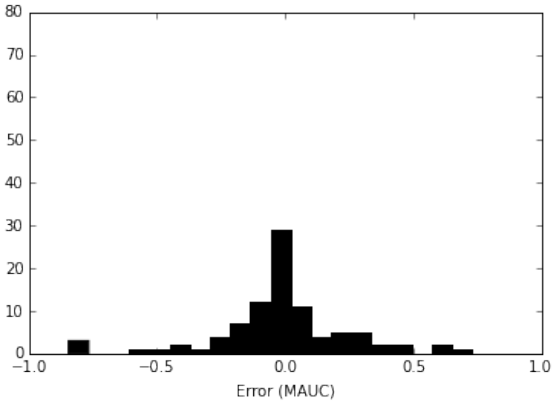
(a) MLP actual versus predicted MAUC improvement



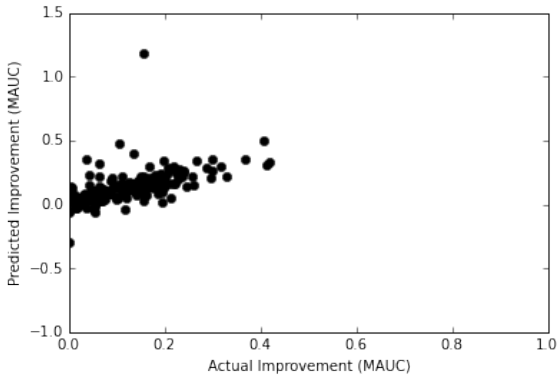
(b) MLP error distribution



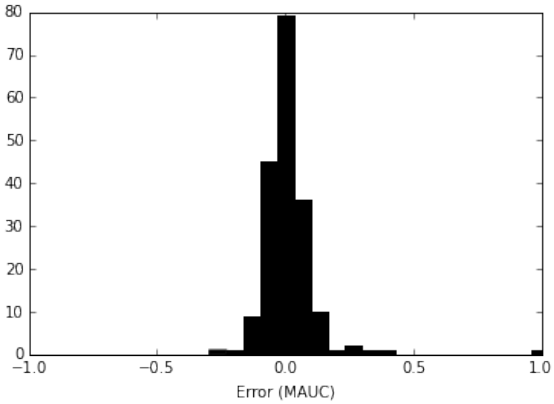
(c) SVM actual versus predicted MAUC improvement



(d) SVM error distribution



(e) Decision Tree actual versus predicted MAUC improvement



(f) Decision Tree error distribution

Figure 4.4: Meta-learner performance results for predicting MAUC improvement after hyper-parameter optimization.

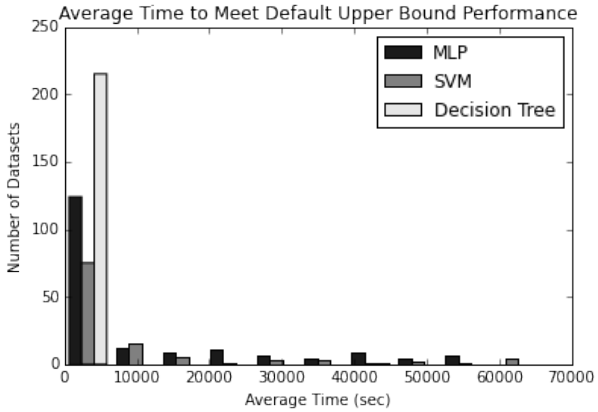


Figure 4.5: Time to meet default upper bound

Chapter 5

Conclusion

Initially we set out to test the hypothesis that hyper-parameter optimization does not yield improved models for a significant amount of datasets, since in [14] they found that about 20% of the time hyper-parameter optimization resulted in no improvement over default hyper-parameters. Based on our experiments that is not the case. In almost every instance there was improvement over default hyper-parameters after hyper-parameter optimization except in the cases where default hyper-parameters yielded a perfect result.

Our experimental setup was in many ways different from the setup in [14]. The changes we made from [14] were made so that our results were relevant and useful to machine learning practitioners. We used a different optimization method, a genetic algorithm instead of PSO, we did minimal preprocessing to the datasets whereas in [14] all of the datasets were binarized if they were not already, we used a different performance metric, MAUC instead of AUC, to accommodate those data sets that were multi-label, and we optimized over a larger set of hyper-parameters for each algorithm.

Any of those differences alone or in combination with each other could explain our different results. A significant factor in the differences in results could have been our effort to leave the data sets untouched. Multi-label problems are inherently more challenging than binary-label problems which could have led to us finding significant benefits from hyper-parameter optimization. Also we considered more hyper-parameters for optimization, which may have led to more possible improvement from hyper-parameter optimization due to having more degrees of freedom in our models.

We also found that especially for Decision Tree and MLP algorithms we can predict how much improvement we can expect from hyper-parameter optimization. If the hyper-parameters for the optimization method such as time to optimize, optimization approach, or learning algorithm hyper-parameters are changed, then the model trained for this paper may not be useful. However, it is possible to predict how much improvement can be expected from hyper-parameter optimization with the meta-features used here. This is especially useful to practitioners who would like to get a sense for whether it is worth it to them to invest time in hyper-parameter optimization.

In looking at the amount of time it takes to beat default hyper-parameter results, we found that Decision Tree meets the benchmark quickly, but that SVM and MLP can take significantly longer. SVM has a wider range of time between the average minimum time and the average maximum time to meet default hyper-parameters than MLP, but MLP has a higher average time to meet default hyper-parameter performance. It is difficult to predict the amount of time to meet the default hyper-parameter performance benchmark, however, the classifiers that determined if the default hyper-parameter benchmark could be met within a certain amount of time were much more successful. These models produced by the classifiers would be useful to practitioners assuming they specify the amount of time they have for optimization. They would then be able to determine if it is likely that the optimization algorithm could find hyper-parameters better than the default hyper-parameters in the specified amount of time.

Future research could include repeating this experiment with more learning algorithms to see if the patterns continue. We chose to use some of the most popular algorithms here, but it would be interesting to see how other learning algorithms generally respond to hyper-parameter optimization.

References

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

[2] James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc., 2011.

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

[4] Thomas G Dietterich and Eun Bae Kong. Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical report, Technical report, Department of Computer Science, Oregon State University, 1995.

[5] Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger Hoos, and Kevin Leyton-Brown. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *Proceedings of the NIPS workshop on Bayesian Optimization in Theory and Practice*, pages 1–5, 2013.

[6] David J Hand and Robert J Till. A simple generalisation of the area under the roc curve for multiple class classification problems. *Machine learning*, 45(2):171–186, 2001.

[7] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

[8] Rafael G Mantovani, André LD Rossi, Joaquin Vanschoren, Bernd Bischl, and André CPLF Carvalho. To tune or not to tune: recommending when to adjust svm hyper-parameters via meta-learning. In *Proceedings of the Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–8. IEEE, 2015.

[9] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent

Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[10] Martin Pelikan. Hierarchical bayesian optimization algorithm. *Hierarchical Bayesian Optimization Algorithm*, pages 105–129, 2005.

[11] Parker Ridd and Christophe Giraud-Carrier. Using metalearning to predict when parameter optimization is likely to improve classification accuracy. In *Proceedings of the 2014 International Conference on Meta-learning and Algorithm Selection-Volume 1201*, pages 18–23. CEUR-WS. org, 2014.

[12] Michael R Smith, Logan Mitchell, Christophe Giraud-Carrier, and Tony Martinez. Recommending learning algorithms and their associated hyperparameters. In *Proceedings of the 2014 International Conference on Meta-learning and Algorithm Selection-Volume 1201*, pages 39–40. CEUR-WS. org, 2014.

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

[14] Quan Sun and Bernhard Pfahringer. Pairwise meta-rules for better meta-learning-based algorithm ranking. *Machine learning*, 93(1):141–161, 2013.

[15] 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. ACM, 2013.

[16] 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.

[17] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

Appendices

A Datasets

acute-inflammations	ada_prior	adult
amazon-commerce-reviews	analcatdata_creditscore	annealing
AP_Breast_Colon	AP_Breast_Kidney	AP_Breast_Lung
AP_Breast_Omentum	AP_Breast_Ovary	AP_Breast_Prostate
AP_Breast_Uterus	AP_Colon_Kidney	AP_Colon_Lung
AP_Colon_Omentum	AP_Colon_Ovary	AP_Colon_Prostate
AP_Colon_Uterus	AP_Endometrium_Breast	AP_Endometrium_Colon
AP_Endometrium_Kidney	AP_Endometrium_Lung	AP_Endometrium_Omentum
AP_Endometrium_Ovary	AP_Endometrium_Prostate	AP_Endometrium_Uterus
AP_Lung_Kidney	AP_Lung_Uterus	AP_Omentum_Kidney
AP_Omentum_Lung	AP_Omentum_Ovary	AP_Omentum_Prostate
AP_Omentum_Uterus	AP_Ovary_Kidney	AP_Ovary_Lung
AP_Ovary_Uterus	AP_Prostate_Kidney	AP_Prostate_Lung
AP_Prostate_Ovary	AP_Prostate_Uterus	AP_Uterus_Kidney
ar4	ar6	arcene
arrhythmia	audiology	australian
auto	autoUniv-au1-1000	autoUniv-au4-2500
autoUniv-au6-1000	autoUniv-au6-400	autoUniv-au6-750
autoUniv-au7-1100	autoUniv-au7-500	autoUniv-au7-700
backache	balance-scale	banana
banknote-authentication	baseball	biomed
blogger	blood-transfusion	breast-cancer-w-diag
breast-cancer-w-orig	breast-cancer-w-prog	breast-cancer
breast-tissue	bridges	car
cast-metal1	chess-kr-vs-kp	click-prediction-small
climate-model-simulation-crashes	cnae-9	colleges_aaup
congressional-voting-records	connect-4	connectionist-bench-sonar
connectionist-bench-vowel	contraceptive-method-choice	covtype
credit-approval	credit-g	cylinder-bands
dermatology	diabetes	diggle_table_a2
dresses-sales	echocardiogram	ecoli
eeg-eye-state	eucalyptus	fertility
first-order-theorem-proving	gas-drift	glass-identification
grub-damage	haberman-survival	hayes-roth
heart-disease-cleveland	heart-disease-hungarian	heart-disease-switzerland
heart-disease-va	heart-long-beach	hepatitis
hill-valley	horse-colic	image-segmentation
internet-ads	ionosphere	iris
irish	kc2	kropt
letter-recognition	liver_disorders	lsvt
lymphography	magic-gamma-telescope	magic-telescope
mammographic_mass	mammography	mc1

mc2	meta-data	molecular-biology-splice-junction
molecular-biology_promoters	monks-problems-1	monks-problems-2
monks-problems-3	mushroom	mw1
nomao	nursery	oil_spill
one-hundread-plants-margin	one-hundread-plants-shape	one-hundread-plants-texture
optical-recognition	OVA_Breast	OVA_Colon
OVA_Endometrium	OVA_Kidney	OVA_Lung
OVA_Omentum	OVA_Ovary	OVA_Prostate
OVA_Uterus	ozone-level-8hr	page-blocks
parkinsons	pc1	pc2
pc3	pc4	pen-basedi-recognition
phoneme	pie-chart1	pie-chart2
pie-chart3	pie-chart4	pima-indians-diabetes
pizza-cutter1	pizza-cutter3	planning-relax
poker-hand	popular-kids	primary-tumor
profootball	qsar-biodeg	qualitative-bankruptcy
rmftsa_sleepdata	robot-failures-lp4	robot-failures-lp5
sa-heart	satimage	schizo
seeds	segment-challenge	segment
seismic-bumps	semeion	servo
skin-segmentation	sonar	soybean-large
spambase	spect-heart	statlog-german-credit
statlog-heart	statlog-landsat	statlog-shuttle
statlog-vehicle	steel-plates-fault	tamilnadu-electricity
teaching-assistant-eval	thoracic-surgery	thyroid-disease-allhypo
thyroid-disease-sick	tic-tac-toe	unbalanced
user-knowledge	vertebra-column	volcanoes-a1
volcanoes-a2	volcanoes-a3	volcanoes-a4
volcanoes-b1	volcanoes-b2	volcanoes-b3
volcanoes-b4	volcanoes-c1	volcanoes-d1
volcanoes-d2	volcanoes-d3	volcanoes-d4
vote	vowel	walking-activity
wall-robot-navigation	waveform-5000	wilt
wine	yeast	zoo

B Meta-features

No.	Meta-feature	Description
1	classes	Number of classes
2	attributes	Number of attributes
3	numeric	Number of numeric attributes
4	nominal	Number of nominal attributes
5	samples	Number of samples
6	dimensionality	attributes / samples
7	numeric_rate	Proportion of numeric attributes
8	nominal_rate	Proportion of nominal attributes
9	symbols_min	Minimum number of nominal attribute values
10	symbols_max	Maximum number of nominal attribute values
11	symbols_mean	Mean number of nominal attribute values
12	symbols_sd	Standard deviation of the number of nominal attribute values
13	symbols_sum	Total number of nominal attribute values
14	class_prob_min	Percentage of elements in the minority class
15	class_prob_max	Percentage of elements in the majority class
16	class_prob_mean	Average number of elements by class
17	class_prob_sd	Standard deviation of elements by class
18	skewness	Lack of symmetry over numeric attributes
19	skewness_prep	Lack of symmetry over normalized numeric attributes
20	kurtosis	A measure of the peakedness of the dataset over numeric attributes
21	kurtosis_prep	A measure of the peakedness of the dataset over normalized numeric attributes
22	abs_cor	Average absolute correlation between attributes
23	cancor_1	Canonical correlation between labels and attributes
24	fract_1	1-D variance fraction coefficient
25	class_entropy	Class entropy
26	normalized_class_entropy	Normalized class entropy
27	attribute_entropy	Attribute entropy
28	normalized_attribute_entropy	Normalized attribute entropy

No.	Meta-feature	Description
29	joint_entropy	Joint entropy of class and attribute
30	mutual_information	Mutual information of class and attribute
31	equivalent_attributes	Equivalent number of attributes
32	noise_signal_ratio	Noise-signal ratio (amount of irrelevant information in a dataset)
33	nodes	Number of nodes in an induced Decision Tree
34	leaves	Number of leaves in an induced Decision Tree
35	nodes_per_attribute	Ratio of the number of tree nodes to the number of attributes in an induced Decision Tree
36	nodes_per_instance	Ratio of the number of tree nodes to the number of instances in an induced Decision Tree
37	leaf_corrobation	Average strength of support of each tree leaf
38	level_min	Minimum number of nodes at one level in an induced Decision Tree
39	level_max	Maximum number of nodes at one level in an induced Decision Tree
40	level_mean	Average number of nodes on a level in an induced Decision Tree
41	level_sd	Standard deviation of the number of nodes on a level in an induced Decision Tree
42	branch_min	Length of the shortest branch in an induced Decision Tree
43	branch_max	Length of the longest branch in an induced Decision Tree
44	branch_mean	Average branch length in an induced Decision Tree
45	branch_sd	Standard deviation over branch lengths in an induced Decision Tree
46	attribute_min	Minimum number of occurrences of an attribute in an induced Decision Tree
47	attribute_max	Maximum number of occurrences of an attribute in an induced Decision Tree
48	attribute_mean	Average number of occurrences of an attribute in an induced Decision Tree

No.	Meta-feature	Description
49	attribute_sd	Standard deviation over number of occurrences of an attribute in an induced Decision Tree
50	naive_bayes	Accuracy of the Nave Bayes algorithm
51	lda	Accuracy of LDA
52	stump_min	Minimum stump accuracy
53	stump_max	Maximum stump accuracy
54	stump_mean	Average stump accuracy
55	stump_sd	Standard deviation over stump accuracies
56	stump_min_gain	Minimum gain ratio over stumps
57	stump_random	Random stump sample
58	nn_1	Accuracy of 1-nn
59	nn_sd	Standard deviation over knn's
60	tree_time	Time to build Decision Tree
61	naive_bayes_time	Time to run Nave Bayes
62	lda_time	Time to run LDA
63	stump_time	Time to run stump
64	nn_time	Time to run k-nearest neighbor
65	simple_time	Time to calculate metafeatures 1-17
66	statistical_time	Time to calculate metafeatures 18-24
67	inftheo_time	Time to calculate metafeatures 25-32
68	total_time	Time to calculate metafeatures 1-64