
# A METAMODEL RECOMMENDATION SYSTEM USING META-LEARNING

THESIS

Megan K. Woods, CTR

AFIT-ENS-MS-20-M-182

**DEPARTMENT OF THE AIR FORCE**
**AIR UNIVERSITY**

**AIR FORCE INSTITUTE OF TECHNOLOGY**

**Wright-Patterson Air Force Base, Ohio**

AFIT-ENS-MS-20-M-182

# A METAMODEL RECOMMENDATION SYSTEM USING META-LEARNING

THESIS

Presented to the Faculty

Department of Operational Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Operations Research

Megan K. Woods, BS

CTR

March 2020

AFIT-ENS-MS-20-M-182

# A METAMODEL RECOMMENDATION SYSTEM USING META-LEARNING

Megan K. Woods, BS

CTR

Committee Membership:

Dr. J. Weir
Chair

Lt Col B. Cox
Member

AFIT-ENS-MS-20-M-182


## Abstract

The importance and value of statistical predictions increase as data grows in availability and quantity. Metamodels, or surrogate models, provide the ability to rapidly approximate and predict information. However, selection of the appropriate metamodel for a given dataset is often arduous, and the choice of the wrong metamodel could lead to considerably inaccurate results. This research proposes and tests the framework for a metamodel recommendation system. The implementation allows for virtually any dataset and preprocesses data, calculates meta-features, evaluates the performance of various metamodels, and learns how the data behaves via meta-learning, thus preparing and bettering itself for future recommendations. Testing on over 500 widely varied datasets, the framework provides positive results, often recommending a metamodel with similar performance as the actual best metamodel.

*I would like to thank my family and friends for their support, encouragement, and understanding as I pursued this program and my research. I would also like to express my incredible gratitude and appreciation to my faculty advisor, Dr. Weir, for his invaluable mentorship and guidance. Not only did he provide me with a skill set in research and academic thinking and writing, he also showed me encouragement and understanding, increased my confidence as an analyst, and inspired me to do great work.*

# Table of Contents

Abstract ..... iv
Table of Contents ..... vi
Table of Figures ..... vii
List of Tables ..... viii
I. Introduction ..... 1
II. Literature Review ..... 4
2.1 Overview ..... 4
2.2 Metamodel Recommendation Systems ..... 4
2.3 Metamodeling ..... 6
2.4 Data Preparation ..... 10
2.5 Training and Testing Sets ..... 25
III. Methodology ..... 26
3.1 Overview ..... 26
3.2 Datasets ..... 26
3.3 Data Preparation ..... 27
3.4 Meta-features ..... 40
3.5 Training and Testing Sets ..... 42
3.6 Metamodels ..... 42
3.7 Proposed Framework ..... 43
3.8 Validation Run ..... 45
IV. Analysis and Results ..... 47
V. Conclusions and Recommendations ..... 53
Appendix A ..... 57
Bibliography ..... 58

# Table of Figures

FIGURE 1. SCHEMATIC DIAGRAM OF RICE'S MODEL WITH SELECTION BASED ON PROBLEM FEATURES (RICE, 1976) ..... 2

FIGURE 2. METAMODEL RECOMMENDATION SYSTEM FRAMEWORK OFFERED BY SAVCHENKO ET. AL..... 5

FIGURE 3. HIERARCHY OF DATATYPES..... 15

FIGURE 4. ADJUSTED HIERARCHY OF DATATYPES ..... 16

FIGURE 5. HISTOGRAMS OF CAR MAKERS IN NOTIONAL DATASET ..... 22

FIGURE 6. FLOWCHART FOR DETERMINING DATATYPES ..... 36

FIGURE 7. PROPOSED FRAMEWORK FOR METAMODEL RECOMMENDATION SYSTEM..... 43

FIGURE 8. DISTRIBUTION OF RELATIVE PERFORMANCES..... 51

# List of Tables

TABLE 1. DATATYPES AND THEIR DEFINITIONS ..... 15

TABLE 2. ADJUSTED DATATYPES AND THEIR DEFINITIONS ..... 17

TABLE 3. SAMPLE OF THE *TOP GEAR* DATASET ..... 27

TABLE 4. CLASSIFICATION RESULTS FOR *TOP GEAR* DATASET ..... 37

TABLE 5. CLASSIFICATION RESULTS FOR *BASEBALL* DATASET ..... 38

TABLE 6. PREPROCESSING METHODS USED PER DATATYPE ..... 39

TABLE 7. COMPARISON OF ORIGINAL *TOP GEAR* DATASET AND DATASET WITH DUMMY VARIABLES ..... 40

TABLE 8. SAMPLE OF META-FEATURES AND NRMSES OF *TOP GEAR* DATASET ..... 45

TABLE 9. EXAMPLE OF LEAVE-ONE OUT METHOD FOR *TOP GEAR* DATASET ..... 46

TABLE 10. PERCENTAGE OF RECOMMENDATIONS WHOSE NRMSE FALL WITHIN AN EPSILON PERCENTAGE OF THE BEST METAMODEL'S NRMSE ..... 48

TABLE 11. PERCENTAGE OF RECOMMENDATIONS WHOSE NRMSE FALL WITHIN A DECIMAL VALUED EPSILON OF THE BEST METAMODEL'S NRMSE ..... 49

TABLE 12. PYTHON MODULES ..... 57

**A METAMODEL RECOMMENDATION SYSTEM USING METALEARNING**

# I. Introduction

The world of statistical predictions increases in importance and value as data becomes more readily accessible and available in large quantities. In its most basic sense, the standard prediction requires some knowledge of past patterns and outcomes to provide a plausible forecast. Information can be modeled through some type of algorithm, or metamodel, which can then be used to make predictions. The term "meta" signifies "a superstructure over some object" (Savchenko & Stepashko, 2018). Therefore, a metamodel, or surrogate model, is a model of a model. Metamodels are data-driven and use their input data to provide rapid approximations of the system they aim to represent.

Countless metamodels are in existence, all with their own unique abilities and pros, as well as limitations and cons. Much research has been done to compare metamodels and their ability to perform and provide accurate results. A metamodel's superiority to others depends on many factors, such as parameters used to tune the metamodel, as well as information regarding the input data (e.g., dimensions and distributions of the input data, datatypes, and other underlying mathematical properties). A single best metamodel has not been found. Given this, there is a need to determine the appropriate metamodel for a given dataset. Often referred to as the algorithm selection problem, the chosen metamodel does not always produce the perfect representation of the dataset (nor should it be required to do so). The outcome of a recommendation system depends greatly on the selection pool and input data. Selection of the "runner up" can still provide adequate modeling results and

may even prove better in other performance categories, such as computational expense and number of parameter specifications.

Whereas metamodeling itself is not a new concept, the algorithm selection problem has recently surfaced within the literature. Published in 1975, Rice's paper "The algorithm selection problem" brought forth a structure for designing an algorithm selection system. The framework comprises four characteristics with mappings between the different spaces: (1) the problem space, comprised of the dataset instance; (2) the feature space, comprised of features of the problem; (3) the algorithm space, containing the pools of algorithms from which to select a good or best model; and (4) the performance measure, which evaluates each algorithm's performance (Rice, 1976). A schematic diagram of this framework is shown in Figure 1.

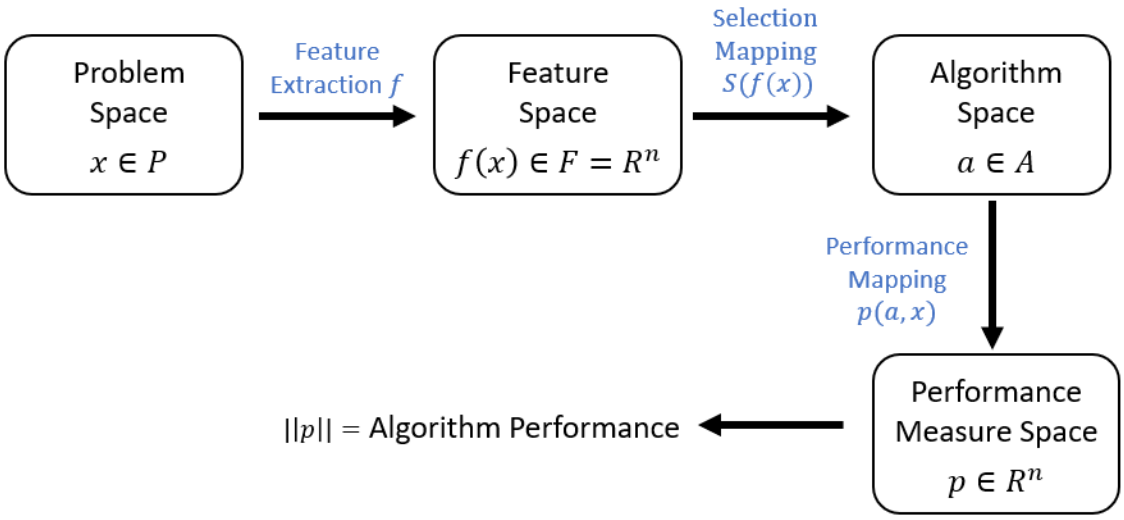


**Figure 1. Schematic diagram of Rice's model with selection based on problem features (Rice, 1976)**

The purpose of the framework is to recommend a metamodel for the given dataset, given its features. Ideally, a new dataset with similar features as another previously run

dataset would perform similarly in terms of performance measures for the various algorithms and would, therefore, receive the same metamodel recommendation as its counterpart.

A more recent paper, “A recommendation system for meta-modeling: A meta-learning based approach”, proposes a metamodel recommendation system for “well-behaved data” – data that is generated using mathematical functions, where the inputs directly determine the outputs (Cui, Hu, Weir, & Wu, 2016). Although this framework has been proven to work well for “nice” data, it requires tuning to be able to function with real-world data. To adapt to non-theoretical datasets, this research extends the framework presented by Cui et. al by automating the selection of a good algorithm for any type of dataset, rather than limiting it to mathematical functions. This research aims to expose if the proposed metamodel recommendation system can get exceptional results when inputting data of different sources and with different relationships.

# II. Literature Review

## 2.1 Overview

The purpose of this chapter is to present previous work related to this study, all of which contribute to the overall flow and elegance of the proposed metamodel recommendation system. This chapter first offers previous work on similar frameworks that inspired the creation of this research. It then presents general information on the metamodels of choice, followed by specific data preparation techniques that inform the main contribution of the proposed structure. Lastly, it explains the process of model validation using a training and testing split.

## 2.2 Metamodel Recommendation Systems

This research revealed several papers and previous studies that use and evaluate metamodel recommendation systems. In their paper, "A recommendation system for meta-modeling: A meta-learning based approach", Cui et al. propose a generalized framework for a metamodel recommendation system (Cui, Hu, Weir, & Wu, 2016). They use Latin hypercube sampling to determine a representative dataset from 44 benchmark functions. After computing 15 meta-features, they reduce these features using three feature reduction methods and then fit them using six different metamodels. Accuracy is measured using normalized root mean squared error (NRMSE). Their implementation achieves 94% correlation on rankings, as determined by Spearman's rank correlation coefficient, and 91% accuracy on the best predicted model using a hit ratio. Their paper's main contributions include (1) creating the framework for the metamodel recommendation system for simulation, (2) improving computational efficiency by reducing it from an order

of minutes experienced by traditional approaches to an order of seconds, (3) suggesting a comprehensive set of meta-features and feature reduction techniques, and (4) validating their framework using benchmark cases.

In their paper, “Short-term building energy model recommendation system: A meta-learning approach”, Cui et al. implement a metamodel recommendation system for the specific case of forecasting energy building profiles (Cui, Wu, Hu, Weir, & Li, 2016). They use building characteristics and statistical and time series meta-features to identify appropriate forecasting models for specific energy buildings.

In their paper, “Metamodeling as a Way to Universalization of Inductive Modeling Tools”, Savchenko et al. offer the framework for a metamodel recommendation system. Their paper indicated the intent to create an automated process, although they point out that tasks could also be user-interactive, such as the final selection of the metamodel. Their process is presented in Figure 2.

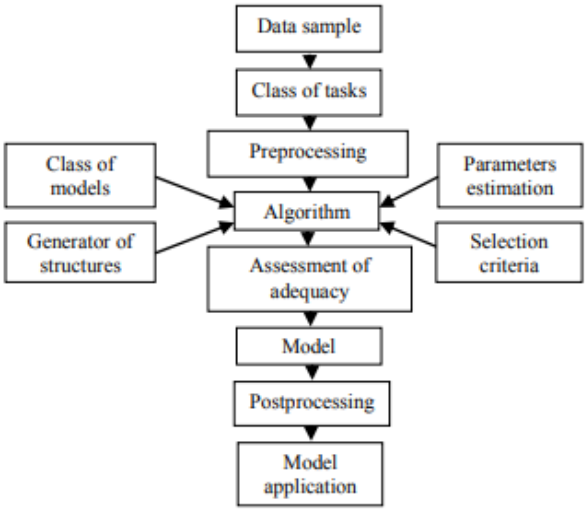


**Figure 2. Metamodel recommendation system framework offered by Savchenko et. al**

One element of their recommendation system, and a crucial component of ours, is the suggested implementation of a preprocessing step, which would include "data visualization, imputing or eliminating omissions, filtering noise in data, checking the homogeneity of a sample, normalizing data", and the like (Savchenko & Stepashko, 2018). They also propose the requirement of the system to determine the needs of a specific dataset, prior to recommendation, to include specifying between continuous and discrete, static and dynamic, linear and nonlinear, one-dimensional and multi-dimensional, stationary and nonstationary, as well as oscillatory and cyclic (Savchenko & Stepashko, 2018).

### 2.3 Metamodeling

As previously mentioned, there are numerous metamodeling techniques, each which bring different capabilities. In his post on R-Bloggers, Piccini lists 101 different machine learning algorithms (Piccini, 2019). Those used in Cui's papers include Kriging, or Gaussian process regression, support vector regression, radial basis function, multivariate adaptive regression splines, artificial neural networks, multilayer perceptron, Bayesian neural networks, generalized regression neural networks, K-nearest neighbor regression, and CART regression trees (Cui, Hu, Weir, & Wu, 2016; Cui, Hu, Weir, & Wu, 2016). Determining the "best" subset of metamodels is beyond the scope of this research. Therefore this research is limited to a sample of six metamodels for testing. The chosen metamodels include multiple linear regression, ridge regression, Bayesian ridge regression, decision tree regression, k-nearest neighbors (KNN), and stochastic gradient descent (SGD). Now each metamodel is briefly explained.

### *2.3.1 Multiple Linear Regression*

Multiple linear regression models the relationship between two or more predictor variables and a response variable, typically via the method of least squares, which aims to minimize the residual sum of squares between the observed and predicted data. The multiple linear regression equation is as follows:

$$\hat{y} = b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_p x_p \tag{1}$$

where $p$ represents the number of predictor variables, $b_0$ is the value of $\hat{y}$ when all independent variables are equal to zero; $b_0, b_1, \ldots, b_p$ are the estimated coefficients of the predictor variables $x_1, \ldots x_p$, respectively; and $\hat{y}$ is the predicted response. Hoerl et al. present the canonical form for multiple linear regression which is shown below:

$$\mathrm{X} = \mathrm{X}^{*}\mathrm{P} \tag{2}$$

and

$$Y = \mathrm{X}^{*}\alpha + \varepsilon \tag{3}$$

where P is an orthogonal transformation such that $\mathrm{X}'\mathrm{X} = \mathrm{P}'\Lambda\mathrm{P}$, $\Lambda = \left(\delta_{ij}\lambda_i\right)$ is the matrix of eigenvalues of $\mathrm{X}'\mathrm{X}$, and $\alpha = \mathrm{P}\beta$, $(\mathrm{X}^{*})'(\mathrm{X}^{*}) = \Lambda$, and $\alpha'\alpha = \beta'\beta$ .

### *2.3.2 Ridge Regression*

Ridge regression is used “to control the inflation and general instability associated with the least squares estimates”, and is known as “weight decay” in neural networks (Hoerl & Kennard, 1970; Bishop, 2006). This technique uses the following equation to estimate the variable coefficients:

$$\hat{B}^{*} = [\mathrm{X}'\mathrm{X} + k\mathrm{I}]^{-1}\mathrm{X}'Y, k \geq 0 \tag{4}$$

To compare ridge regression with multiple linear regression, see the following form as defined by Hoerl et al.:

$$\alpha^* = [(\mathrm{X}^*)'(\mathrm{X}^*) + \mathrm{K}]^{-1}(\mathrm{X}^*)'Y \tag{5}$$

where $\mathrm{K} = \left(\delta_{ij}k_i\right)$ and $k_i \geq 0$.

Bishop notes that regularization, which adds a penalty term to the error function, can be used to control over-fitting (Bishop, 2006). For example, typically the goal is to minimize the error term, expressed often as

$$E(w) = \sum_{n=1}^{N} \{y(x_n, \boldsymbol{w}) - t_n\}^2 \tag{6}$$

where $y(x_n, \boldsymbol{w})$ are predictions for data point $x_n$ and $t_n$ are the target values.

The error term of ridge regression uses the following form, adding only a single penalty term to (6):

$$E(w) = \sum_{n=1}^{N} \{y(x_n, \boldsymbol{w}) - t_n\}^2 + \frac{\lambda}{2} w^T w \tag{7}$$

The coefficient $\lambda$ determines the regularization term's importance in comparison with the importance of the sum-of-square error term.

### *2.3.3 Bayesian Ridge Regression*

Bayesian regression techniques allow for the use of regularization parameters. Given by Bishop, the specific case of Bayesian ridge regression has a spherical Gaussian prior for the parameter $\alpha$:

$$p(w|\alpha) = \mathcal{N}(w|0, \alpha^{-1}\mathrm{I}) \tag{8}$$

The posterior distribution over $w$ is given by

$$p(w|t) = \mathcal{N}(w|m_N, S_N) \tag{9}$$

where $m_N = \beta S_N \Phi^T t$ and $S_N^{-1} = \alpha \mathrm{I} + \beta \Phi^T \Phi$. Bishop notes that the "[r]egularization of this posterior distribution with respect to $w$ is…equivalent to the minimization of the sum-of squares error function with the addition of a quadratic regularization term". This corresponds to (7), with $\lambda = \alpha/\beta$ , making this algorithm very similar to the classical ridge regression (Bishop, 2006).

### *2.3.4 Decision Tree Regression*

Decision tree regression partitions a dataset into subsets by creating decision points that place the data into different nodes (Acharya, Armaan, & S, 2019). Regression trees are built top-down. The root node is chosen to be the decision point that minimizes the sum of the residuals. Each subsequent decision node is then chosen in the same manner. A stopping criterion, such as the minimum number of observations in a node, determines when a node becomes an endpoint, or leaf. This leaf contains a numeric value that represents the output value.

### *2.3.5 k-Nearest Neighbors*

K-nearest neighbors regression (KNN) determines which $k$ datapoint's input data are closest to the inputs of a test datapoint, in terms of Euclidean distance. The distance $d$ between training point $p$ and testing point $q$, each with $n$ inputs, is shown below.

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{10}$$

The $k$ datapoints with the smallest distances are then selected as the nearest neighbors. The mean value of the target variable of these neighbors are then used to predict the target

variable value of the test datapoint. The mean calculation is shown below, where $y(p_j)$ is the response value of neighbor $p_j$ and $\hat{y}(q)$ is the predicted value of target point $q$.

$$\hat{y}(q) = \frac{1}{k}\sum_{j=1}^{k} y(p_j) \tag{11}$$

### *2.3.6 Stochastic Gradient Descent*

To predict the value of a target variable, stochastic gradient descent (SGD) implements the following minimization algorithm:

$$\theta_t^* = \arg\min_\theta \frac{1}{t}\sum_{i=1}^{t} (L(f_\theta(x_i), y_i) + R(\theta)) \tag{12}$$

where there are $t$ datapoints, $\theta_t^*$ is the parameter that minimizes empirical cost, $(x_i, y_i)$ is training point $i$, $L(s, y)$ is a loss function (which provides a small value if $s$ is a good prediction for $y$), and $R(\theta)$ is a regularization function (Xu, 2011). The term $f_\theta(x_i)$ is the function that predicts the outcome variable $y_i$ given the observed variable $x_i$. SGD is a common algorithm to solve (12), in which the sequence $\theta_n$ approximates $\theta_t^*$ and is updated iteratively through:

$$\theta_t = \theta_{t-1} - \gamma_t g(\theta_{t-1}, d_t) \tag{13}$$

where $\gamma_t$ is the learning rate at step $i$, $\gamma_t$ is either a scalar or a matrix, $g(\theta, \xi) = \frac{\partial L(\theta, d)}{\partial \theta}$ is the gradient of the loss function, and $D_t = (d_1, \dots, d_t)$ are the training samples at step $t$ (Xu, 2011).

## 2.4 Data Preparation

One component absent from Rice's paper, as presented in Chapter I, and Cui's papers is data preparation prior to metamodeling. This step is present in Savencho's

suggested framework and plays a critical role in the execution of this research. Rice's paper focuses on collections of functions (e.g., selecting a program to solve ordinary differential equations) and does not call for data preparation, engineering, or any of the like. Similarly, the framework proposed by Cui et al. tests on 44 functions, meaning that each problem has underlying mathematical properties and, therefore, do not require any cleaning prior to making predictions (Cui, Hu, Weir, & Wu, 2016). The test data used for research in "Short-term building energy model recommendation system" considers simulated data and does not mention any data preparation procedures (Cui, Wu, Hu, Weir, & Li, 2016). Unlike the previously mentioned papers on recommendation systems, Savencho's work proposes a system that allows for both pre- and post-processing. Because the recommendation framework allows for any type of tabular dataset, and, therefore, does not guarantee that the data will be ready for use, this research implements a preprocessing step as well.

Many machine learning techniques require data preparation to be considered of any use. It is not always clear how to handle troublesome information, such as missing values, categorical inputs, and unscaled data. This raises the need for data cleaning and reformatting. Detailed information on data cleansing processes can be found in many textbooks and courses relating to machine learning. This chapter goes through the main preprocessing steps and describes common techniques and their alternatives. Among these include how to handle missing data, determining information to maintain or remove, how to handle categorical data, scaling techniques, as well as several additional steps that can be implemented.

### *2.4.1 Missing Data*

One major issue encountered when preprocessing is how to handle missing data. The two main approaches are to (1) remove observations or fields containing any missing values, and (2) replace the missing value with another value. This topic has been researched extensively, but no one method is considered the best. Several approaches include replacing missing values with one of the following (Larose & Larose, 2014):

- A value deduced by considering the context (e.g., a missing value could imply an input of 0)
- An analyst-specified constant
- The mean or mode of the variable
- A random value generated from the observed data's distribution
- A prediction (or label) based on multiple linear regression or stochastic regression (or a classification algorithm)
- The value from another observation with similar inputs for a subset of variables (referred to as "hot deck imputation")
- An analyst-determined value based on results from multiple imputation (an approach that blends classical and Bayesian statistics and offers several potential imputed values. The analyst "specifies an imputation model, imputes several data sets, analyzes them separately, and then combines the results" (Ette & Williams, 2007)).

The previously mentioned methods for handling missing data are not without their faults. As mentioned in "Discovering Knowledge in Data", omitting observations or fields containing missing inputs may be wasteful. Even more detrimental is the possibility that

this could lead to a biased subset of data (Larose & Larose, 2014). For example, a dataset could have missing values due to patients dropping from medical treatment prior to completion. This could potentially remove a population from the dataset that each responded positively (or negatively) to the treatment – information that would not be captured upon their removal. Mean (or mode) imputation "decreases variability between individuals and biases correlations with other variables", both of which are unfavorable aftermaths (Ette & Williams, 2007). Furthermore, a downfall to many of these methods is that the imputed data is treated with certainty. A multiple imputation approach looks to remove this uncertainty by considering several possible inputs. The main flaw of multiple imputation is its computational expense (Ette & Williams, 2007), which may not be a major setback, given the advanced power of machines today.

### *2.4.2 Filtering*

The following paragraphs deal with questionable data that compel the decision to either maintain them within the dataset or filter them out. The first concerns unary variables (those that have only one level of input). Logic proposes to remove these variables, as identical inputs for all observations contribute no information. This segues into the topic of "nearly" unary variables, in which a large percentage of the data have the same input. Larose suggests removing nearly unary variables, with the requirement that the analyst determine the threshold to be considered nearly unary (e.g., nearly unary variables are those that have, say, 99% of their data having the same input). Larose warns against removing a variable simply because a large proportion has missing values or because it is highly correlated with other variables, as doing so could still discard important information (Larose & Larose, 2014).

Another area of consideration is whether to filter out duplicate observations. Repeat observations could be present because (1) the record was inadvertently copied twice, or (2) two or more observations had the exact same inputs. In the first scenario, the record should be deleted, as it overweighs those inputs. However, the second scenario should be maintained to support the fact that that combination of inputs occurred on more than one occasion. Larose suggests using common sense when addressing this issue. To use their example, a dataset with three nominal fields, each with three levels, would generate $3^3 = 27$ possible combinations. A dataset containing more than 27 observations would undoubtfully result in a repeated combination (Larose & Larose, 2014).

### *2.4.6 Datatypes*

#### *2.4.6.1 Continuous vs Discrete*

Datatypes play an important role in machine learning, as different datatypes require different preprocessing methods. In testing, the preprocessing recognizes two main datatypes: continuous and discrete. Continuous data are always numeric, whereas discrete are both numeric and non-numeric (i.e., textual). The main difference between the two classes is that continuous data can always increase in precision, whereas discrete data cannot. However, this distinction should not be extended to non-numeric variables. For example, one could argue that a nominal variable representing the type of pet belonging to an individual could be further characterized into breeds, (i.e., made more precise).

#### *2.4.6.2 Subclasses for Datatypes*

Both continuous and discrete datatypes have subclasses, which are adopted from Valera et al. and are listed as follows: continuous data includes real-valued, positive real-valued, and interval data; and discrete data includes categorical, binary, ordinal, and count

data (Valera & Ghahramani, 2017). These subtypes are captured in the hierarchy presented in Figure 3.

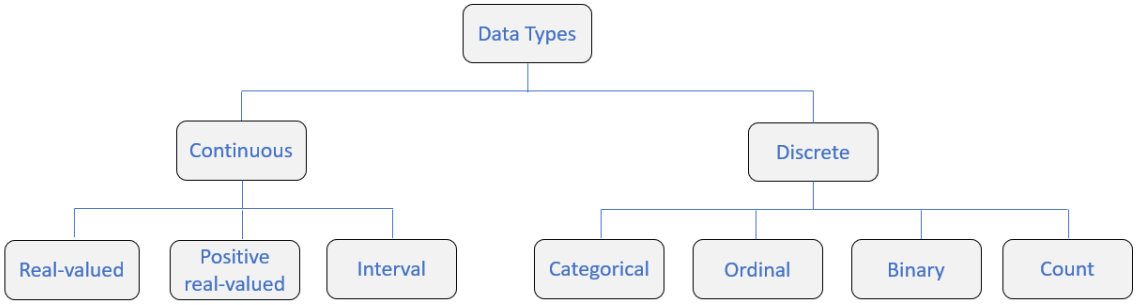


**Figure 3. Hierarchy of datatypes**

Each datatype is listed in Table 1, along with their parent class and definition, as provided by Valera et al. (Valera & Ghahramani, 2017).

**Table 1. Datatypes and their definitions**

| Datatype | Parent Class | Definition |
|---|---|---|
| Real-valued | Continuous | $x \in \mathbb{Z}$ |
| Positive real-valued | Continuous | $x \in \mathbb{Z}^{+}$ |
| Interval | Continuous | $x \in (\theta_L, \theta_U) \subseteq \mathbb{Z}$, where $\theta_L \leq \theta_U$ |
| Categorical | Discrete | $x$ comes from a finite, unordered set |
| Binary | Discrete | $x \in \{0,1\}$ |
| Ordinal | Discrete | $x$ comes from a finite, ordered set |
| Count | Discrete | $x \in \mathbb{N}$ |

*2.4.6.3 Proposal of Datatype Hierarchy*

The above subcategories are all-encompassing – that is, all data falls into one of the datatypes listed in Table 1. However, this research proposes a revised organization of the hierarchy. That is not to say that the one offered by Valera et al. is incorrect, but rather that, for these purposes, another approach is more suitable. The new hierarchy groups together datatypes that are to be handled with the same preprocessing methods. This research only

adjusts the discrete variables, where discrete variables are subcategorized into categorical and count variables. Categorical variables are then decomposed into nominal, ordinal, and binary variables. These variables represent finite data, whereas count variables could, theoretically, take on infinite values. In the adjusted hierarchy, ordinal and binary variables maintain their previous definitions, whereas nominal data adopts the previous definition of categorical data. Figure 4 shows the modifications.

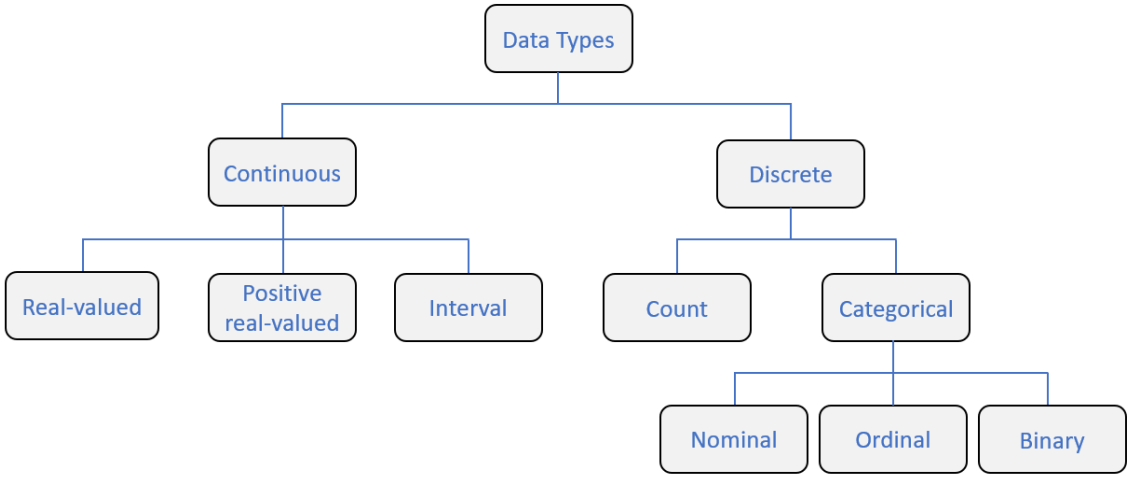


**Figure 4. Adjusted hierarchy of datatypes**

Table 2 shows each datatype, its parent class(es), a definition, and whether it is numeric. One note of ordinal variables is that, although there is an order in which the levels abide by, there is no way to measure the difference between any two levels. Note that this research considers binary to be a numeric datatype, as possible inputs are 0 and 1. A variable that has two textual levels (e.g., “male” and “female”) is considered nominal, although it can be transformed into binary.

**Table 2. Adjusted datatypes and their definitions**

| Datatype | Parent Class(es) | Definition | Numeric |
|---|---|---|---|
| Real-valued | Continuous | $x \in \mathbb{Z}$ | Yes |
| Positive real-valued | Continuous | $x \in \mathbb{Z}^+$ | Yes |
| Interval | Continuous | $x \in (\theta_L, \theta_U) \subseteq \mathbb{Z}$, where $\theta_L \leq \theta_U$ | Yes |
| Nominal | Discrete, Categorical | $x$ comes from a finite, unordered set | No |
| Ordinal | Discrete, Categorical | $x$ comes from a finite, ordered set | Possibly |
| Binary | Discrete, Categorical | $x \in \{0,1\}$ | Yes |
| Count | Discrete | $x \in \mathbb{N}$ | Yes |

*2.4.6.4 Examples of Different Datatypes*

Examples of the different datatypes are now presented, beginning with continuous variables, which include real-valued, positive real-valued, and interval data. For each of the continuous variables, it follows that in between any two values, there is an infinite number of values that are valid inputs. An example of a real-valued variable is the equation $y = x$, as any value on the real number line is a plausible input. An example of a positive real-valued variable is time, because time can take on any positive value. The height of a person is an interval, where possible values fall within the generous range of 0 to 10 feet.

Next, examples of discrete data, beginning with the categorical class, which includes nominal, ordinal, and binary are presented. Several examples of nominal data include the color of traditional M&Ms (i.e., the set {“Green”, “Red”, “Blue”, “Brown”, “Yellow”, “Orange”}), gender (i.e., the set {“Male”, “Female”)}, and types of operating systems (i.e., the set {“Mac”, “Windows”, “Linux”}). Examples of ordinal data include levels of happiness, where possible values come from the set {“Very unhappy”, “Unhappy”, “Okay”, “Happy”, “Very happy”} and number of cylinders in a standard car,

where possible values may include {2, 3, 4, 5, 6, 8}. A binary variable is simply the set {0,1} and measures true/false data, such as whether an individual owns a pet, with 0 representing “no” and 1 representing “yes”. The final datatype is count variables, another subclass of the discrete type. Examples of this datatype include the number of soccer goals scored per game in a season and the number of people at each concert during a band’s tour.

*2.4.6.5 Distinguishing Between Datatypes*

Distinguishing between each subcategory presented above is typically a simple process when completed by a human, although this could still present debate per interpretation. However, more challenges arise when a machine is to automate the categorization of data. The main need is to identify between continuous, ordinal, and count data. As noted in Table 2, both continuous and count data are numeric, and ordinal can possibly be numeric. Identifying a variable as binary (another numeric datatype) is a simple process, as one only needs to verify that the numbers 0 and 1 are the sole levels. Therefore, when presented with numeric data, one must be able to determine which of these categories best describes the variable. It is possible that a continuous variable disguise itself as a discrete variable or vice versa. Likewise, ordinal and count data have the potential to appear identical.

The *Top Gear* dataset, along with the dataset, *Baseball* provide an example of this potential problem. The *Top Gear* dataset comes from the car review television show, *Top Gear*, and provides various attributes of a variety of cars, along with a “Verdict” column that represents the television program’s hosts’ overall opinion of the cars. The variable, “Length”, from the *Top Gear* dataset measures the length of cars in millimeters. As this is a quantitative measurement and because the measurements could become more precise (by

using more decimals or via a more precise form of measurement, such as the micrometer), this variable is continuous. Nevertheless, as any car dataset would do, the *Top Gear* dataset rounds length to the nearest whole number of its utilized unit (millimeters). We now compare the length variable to one from the dataset, *Baseball*, provided by the Python package, *pydataset*. This dataset contains baseball statistics and includes the variable "hits", giving the number of hits in the careers of various baseball players. This statistic is undoubtedly a count variable, as it represents frequency. However, it is possible that the values of this variable mimic those of the length variable in the *Top Gear* dataset. If this is the case, how would a machine automatically distinguish between these two variables? Although the number of hits in the baseball dataset is a count variable, if it appears to be a continuous variable, it should be treated as one in terms of preprocessing.

As another example, imagine a dataset contains the number of goals scored per game during a World Cup. This variable will most likely range from 0 to around 12 (the current record). At the same time, the number of cylinders in a car typically comes from the set {2, 3, 4, 5, 6, 8}. These two variables could appear to be very similar, even though one variable is a count and the other is ordinal. Although these variables are by definition different datatypes, because the numbers within each are very similar (whole numbers ranging from 0 to 10 or from 2 to 8), they should be treated similarly.

*2.4.6.6 Algorithm*

In this research, the recommended method for automating the categorization of continuous and discrete variables is to implement some type of algorithm. Online forums suggest looking at the ratio between the number of unique values in a column and the number of rows in the dataset and comparing that ratio to some threshold. Valera et al.

suggest creating rules based on the number of unique values of a column and the frequency of occurrence of those values (Valera & Ghahramani, 2017). The two examples presented in the previous paragraph lead to an important criterion: rather than correctly identifying the datatype of a variable (where correct identification depends completely on human interpretation), this algorithm's main goal must be to correctly identify which datatype the inputs of a variable *most resemble*, indicating that this variable should be treated as though its machine-classified datatype were in fact the correct datatype. The algorithm to distinguish between the different datatypes is described in Chapter III.

### *2.4.3 Categorical Data*

Categorical data, according to the previously described definitions and hierarchy, can either be ordinal, in which a natural ordering occurs (e.g., unsatisfied, neutral, satisfied); nominal, in which the inputs hold no quantitative value (e.g., North, South, East, West); or binary, where possible values come from the set {0,1}. To be of any use in machine learning algorithms, categorical data must be transformed into some numeric representation.

According to Narsky et al., ordinal variables can be represented by integers "if the relative distance between categorical levels is unimportant", which would be sufficient for an algorithm such as decision trees (Narsky & Porter, 2014). However, many algorithms take distance into account, and so setting categorical levels to an integer would cause an issue in modeling. Therefore, integer values to represent an ordinal variable should be used with caution, as uniform spacing in between numeric values implies uniform spacing between the original categorical variables. For example, if the levels for a variable are "Bad", "Okay", "Good", and "Great" and the associated numeric variables are 1,2,3, and

4, this implies that the difference between “Bad” and “Okay” is the same as that between “Okay” and “Good” since the distance between 1 and 2 and between 2 and 3 are equal. This implication may not, however, be valid.

Nominal data can be transformed using dummy variables, also known as flag variables. This process, referred to as binary encoding, as well as one-hot encoding, is now described. For a $k$-level categorical column, the column is transformed into $k - 1$ columns. The replacement columns are binary to indicate if a particular row originally contained the level represented by the additional column. Of the possible levels, one level does not receive a corresponding dummy column, but is instead represented by having all “0”s in the other levels’ representative columns. Let $N$ be the set of all columns in a given dataset, and let there be $n$ columns in $N$. Let $N_a$ be the set of columns within $N$ that are categorical, where $N_a \subseteq N$ and $A$ is the number of categorical columns. Let $N_b$ be the set of columns in $N$ that are not categorical, where $N_b \subseteq N$ and $B$ is the number of non-categorical columns. Because a column must be categorized as categorical or not categorical, $N_a \cap N_b = \emptyset$ and $N_a \cup N_b = N$. Dummy variable transformation is only performed on columns within $N_a$, (i.e., those that are categorical). Let $k_{n_a}$ be the number of levels in categorical column $n_a$. After binary encoding, the transformed dataset has $B + \sum_{a=1}^{A}(k_{n_a} - 1)$ columns.

An additional step is to create an “other” category when pursuing binary encoding. For example, Figure 5 below shows a variable capturing the frequency of different car makers within a fictional dataset. Standard binary encoding would create eight different variables to represent this categorical variable, as there are nine levels. There are six different makers that each only have between one and three occurrences. Instead of creating

a binary indicator column for each of these makers, they could each be lumped into a combined category. Cars that fall into this "other" category indicate that they are not Ford, Chevrolet, or Toyota. This reduces the additional number of columns from eight to three, which could improve computational runtime and efficiency. This would be a very effective method for variables with many levels, given that those lumped together individually have relatively few occurrences compared to those that maintain their own column.

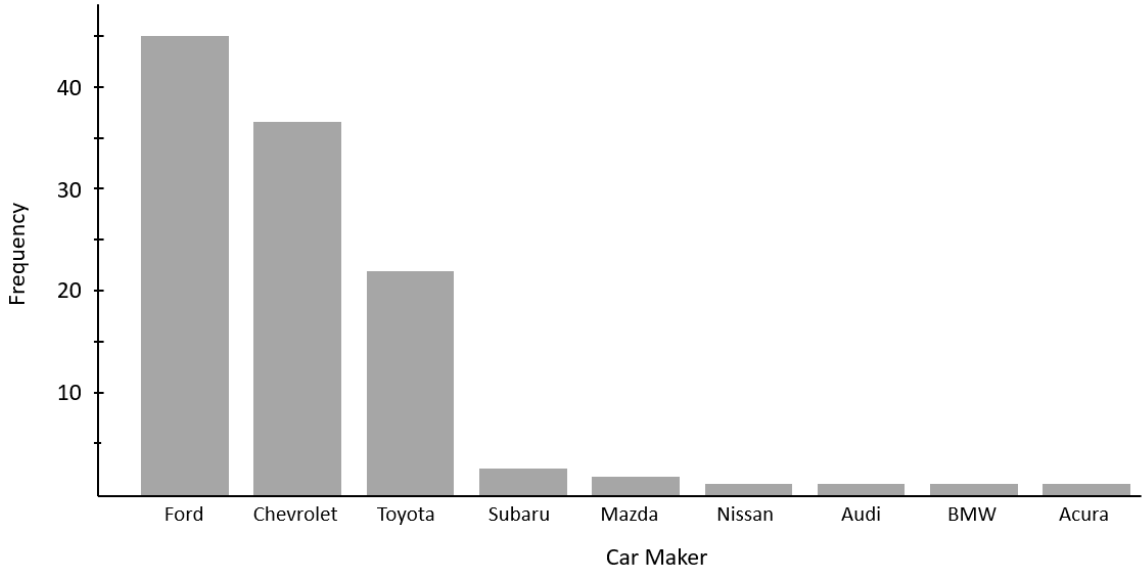


**Figure 5. Histograms of car makers in notional dataset**

### *2.4.4 Scaling*

One very important step in data preparation is scaling the data. Because machine learning algorithms often take magnitude into consideration, while disregarding units, unscaled data causes features with large ranges to have an erroneously greater influence on the results (Larose & Larose, 2014). Furthermore, unscaled data can impact efficiency of learning.

Several options for scaling include the following, each of which are described below.

- Min-max normalization
- Z-score standardization
- Decimal scaling

*2.4.4.1 Min-max Normalization*

The min-max normalization method considers the deviation of the observed value from the absolute minimum value for the given variable and scales this by the range of the variable. This method is represented mathematically by

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{14}$$

where $x'$ is the normalized value, $x$ is the original value, $x_{min}$ is the minimum of the values, and $x_{max}$ is the maximum of the values. This method scales the variables between 0 and 1, so that $x_{min}$ maps to 0 and $x_{max}$ maps to 1.

*2.4.4.2 Z-score Standardization*

Z-score standardization looks at the difference between the observed value and the average value within the column and divides it by the standard deviation of the column's values. The following equation represents this method mathematically:

$$x' = \frac{x - \bar{x}}{\sigma} \tag{15}$$

where $x'$ is the scaled value, $x$ is the original value, $\bar{x}$ is the average, and $\sigma$ is the standard deviation. The resulting scaled variables have a mean of 0 and a standard deviation of 1.

*2.4.4.3 Decimal Scaling*

Decimal scaling ensures that all normalized values fall between $-1$ and $1$ and is represented by the following equation:

$$x' = \frac{x}{10^d} \tag{16}$$

where $x'$ is the scaled value, $x$ is the original value, and $d$ is the number of digits of the input with the largest absolute value. Let $x_{maxAbs}$ be the input with the largest absolute value. The resulting distribution lies in the interval $\left[-\frac{x_{maxAbs}}{10^d}, \frac{x_{maxAbs}}{10^d}\right]$, which falls in between $-1$ and 1.

### *2.4.7 Additional Steps*

The scope of this study does not require any additional data cleansing steps. However, in real-world applications, further preprocessing steps should be considered, to include

- Outlier detection and the potential removal of outliers. Outlier detection serves as an alarm that a datapoint may be an outlier, rather than as a definite conclusion that a given datapoint is an outlier. Therefore, outlier detection requires further investigation to determine whether an outlier is in fact an outlier.
- Removing fields/levels that are no longer relevant or have expired
- Ensuring that inputs make sense regarding their context
- Checking for input errors (e.g., spelling)
- Specifying datatypes for each column (e.g., categorical, count, etc.)
- Splitting columns into separate components (e.g., decomposing time into year, month, day, hour, minute, etc.)
- Feature-reduction techniques (e.g., principal components analysis)

## 2.5 Training and Testing Sets

When building a machine learning algorithm, it is standard to perform a training and testing split to test the predicting abilities of the algorithm. A certain percentage of an observed dataset is labeled as the “training data”, leaving the remaining datapoints to be “testing data”. This enables calculation of how well each metamodel performs on the dataset. The model is built using the training dataset, where the target column is the output. After it is built, the algorithm makes predictions on the testing dataset. Because the original dataset contains the actual values of the target column for the testing dataset, the error in prediction can be determined. There are many articles that reference the ideal splits for specific algorithms, but there is no consensus on the percentages to use for these subsets for any algorithm. Therefore, the implementation uses the default values of the splitting algorithm.

This chapter looked at previous work on recommendation systems, as well as the literature on metamodels and data cleansing techniques. The next chapter uses this literature to guide the development of a metamodel recommendation system.

# III. Methodology

## 3.1 Overview

This research study uses the framework presented by Cui et al. in their article "A recommendation system for metamodeling: A meta-learning based approach" and by Cui et al. in their article "Short-term building energy model recommendation system: A meta-learning approach" (Cui, Hu, Weir, & Wu, 2016; Cui, Wu, Hu, Weir, & Li, 2016). The purpose of this study is to validate whether the proposed metamodel recommendation system framework functions with another application. Rather than using mathematical functions or a specific type of data, this research considers a broad scope of datasets, only requiring the data to be in tabular form. The datasets may or may not have an inherent mathematical relationship, that is, certain combinations of inputs could lead directly to outputs or could be seemingly unrelated in a mathematical sense. Additionally, the collection of datasets could be made up of a combination of those that can be easily represented mathematically, and those that cannot.

## 3.2 Datasets

To make metamodel recommendations, the framework first requires a pool of datasets on which to train. To use this framework, the datasets must be in tabular form, (i.e., each column of a dataset represents an attribute), and each row represents a separate observation. Table 3 shows an example of the *Top Gear* dataset. The columns of this dataset represent different attributes relating to cars, and the rows represent specific cars.

**Table 3. Sample of the *Top Gear* dataset**

| | Maker | Model | Type | Fuel | Price | Cylinders | ⋯ | Verdict | Origin |
|---|---|---|---|---|---|---|---|---|---|
| 0 | Alfa Romeo | Giulietta | Giulietta 1.6 JTDM-2 105 Veloce 5d | Diesel | 21250 | 4 | ⋯ | 6 | Europe |
| 1 | Alfa Romeo | MiTo | MiTo 1.4 TB MultiAir 105 Distinctive 3d | Petrol | 15155 | 4 | ⋯ | 5 | Europe |
| 2 | Aston Martin | Cygnet | Cygnet 1.33 Standard 3d | Petrol | 30995 | 4 | ⋯ | 7 | Europe |
| 3 | Aston Martin | V12 Zagato | V12 Zagato 6.0 V12 Standard 2d | Petrol | 396000 | 12 | ⋯ | 7 | Europe |
| 4 | Aston Martin | Vanquish | Vanquish 6.0 V12 Standard 2d | Petrol | 189995 | 12 | ⋯ | 7 | Europe |
| 5 | Aston Martin | Vantage | V8 Vantage 4.7 V8 420 Standard 2d | Petrol | 84995 | 8 | ⋯ | 8 | Europe |
| 6 | Aston Martin | Vantage Roadster | V8 Vantage 4.7 420 Roadster 2d | Petrol | 93995 | 8 | ⋯ | 7 | Europe |
| 7 | Audi | A1 | A1 1.2 TFSI 86 S line 3d | Petrol | 17025 | 4 | ⋯ | 6 | Europe |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 237 | Volvo | XC70 | XC70 2.0 D3 DRIVe 163 SE Lux 5d | Diesel | 33715 | 5 | ⋯ | 6 | Europe |

To gather a robust collection of datasets, the Python library *pydataset* was used. This package was selected because of its quantity of datasets, the variation among the datasets in terms of size and content, and because the datasets are provided using a tabular format. This package contains an extensive variation of datasets, ranging from those with few observations to those with thousands, and from those containing solely numeric inputs, to those with numeric, binary, and categorical inputs. Datasets from the *pydataset* package, as well as the addition of the *Top Gear* dataset, make up the candidate problem space, as defined in Figure 1. There are 758 problems within the candidate problem space. Throughout the preprocessing steps described below, detailed requirements for transitioning from the candidate problem space into the problem space are discussed.

### 3.3 Data Preparation

As described in Chapter II, preprocessing is nearly always a required step when modeling, as datasets do not often come in a format readily available for testing. Due to the lack of subject-matter expertise on the datasets' contexts, collection methods, and intended purposes, additional preprocessing steps that dive deeper into the actual inputs are forgone. However, in practice, further data cleansing is often needed. For example, the *Top*

*Gear* dataset was inspected and cleaned prior to its use. Upon review of the ranges for numeric attributes, several cars had incorrect inputs, making them erroneous outliers. Fuel efficiencies of 235 and 470 mpg and 0-60 acceleration of 0 seconds were clearly input errors, and cars with these values were, therefore, removed from the dataset. However, in-depth inspection of outlier detection was not accomplished on the datasets provided by pydataset, as this would require much investigation and could rely heavily on prior knowledge of the data at hand. Nevertheless, contextual preprocessing of known datasets should be completed, if possible, and will most likely improve prediction accuracies.

### *3.3.1 Missing Data*

Upon inspection, at least one dataset within the *pydataset* collection contained all missing values. Therefore, in considering the eligibility of a dataset to enter the problem space, any column within a dataset containing all missing values is removed.

In reviewing the options for handling missing data, as detailed in Chapter II, we decided to remove all observations that contained any missing input. Replacing missing values with (1) the mean or mode, or (2) a random value based on the known data's distribution could lead to an impossible observation or one that does not exist. Taking the *Top Gear* dataset as an example and considering option (1) from above, a car with a missing origin would be given an origin of "Europe", as European cars occur most frequently in the dataset. However, this method ignores the inherent relationship between origin and maker, and could erroneously label an Asian car maker, such as Subaru, as a European car. Options (1) and (2) above could potentially create a car that is uncharacteristic of its, say, maker, (e.g., a large Mini Cooper) or one whose inputs do not coincide with one another (e.g., a car with poor torque but great acceleration). These potential outcomes could

possibly be solved via a clustering algorithm or regression, but combinations of other variables could inaccurately play a greater role in determining the input.

In addition to the issues encountered by each method for handling missing data that were mentioned in the literature, several study-specific complications arose regarding the other techniques. The first major cause of concern with selecting an imputation method is the variation among the datasets and that the datasets are unknown. For example, handling missing data for a patient who dropped out of a treatment plan may need to be dealt with completely different than missing data due to a machine malfunction. Again, this would require contextual knowledge of the datasets. Mean or mode imputation or imputation using a clustering or regression algorithm requires that we know the field's datatype. As will later be expanded upon, we use an algorithm to categorize variables as either binary, nominal/ordinal, continuous, or count. As the utilized classification algorithm is not entirely without flaw, a mischaracterization could result in an imputed value that is continuous instead of discrete, or vice versa. Additionally, the approach of replacing missing values with a predefined constant implies that an analyst or subject-matter expert would be deep-diving into the dataset for inspection and contextual analysis (either pre- or post-testing) – a step that was not completed for this study.

Although omitting observations or fields with missing values could lead to a biased subset of data, the potential complications caused by other methods were considered of greater harm than the value of including more information (by using imputation). Further, many data imputation techniques benefit from human-led examination and validation, steps that are not taken in this study.

### *3.3.3 Additional Filtering*

Chapter II introduces the data cleaning technique of removing unary variables and nearly unary variables. For testing, we removed unary variables, but did not remove nearly unary variables, as the threshold for classifying a variable as "nearly" unary would need to be analyst-specified and datasets of different sizes would most likely require different thresholds.

A column that serves as an index column, such as the first column in Table 1, is removed from the dataset. Any columns with textual input that contain all unique values are removed from the dataset, such as the "Model" and "Type" columns from Table 1.

In alignment with the suggestions in Chapter II, we do not remove variables that contain a large portion of missing data, as a threshold would need to be established. Likewise, we do not omit highly correlated variables. It was decided not to remove duplicate observations within a dataset, although, during testing, it was verified that no dataset contained repeated rows.

### *3.3.4 Minimum Number of Numeric Columns – Pre-binary Encoding*

Because the recommendation system is designed to predict metamodels on datasets whose outputs are numeric values, the original datasets (i.e., the dataset prior to binary encoding, which is later explained) must contain at least one column with numeric data. There were 16 datasets that did not satisfy this requirement and, therefore, did not transition from the candidate problem space to the problem space. We do not specify an upper bound on the number of columns within a dataset. After ineligible columns were removed from the dataset, the minimum and maximum number of columns in the qualified datasets is 3

and 212, respectively. Note that these counts (1) include the target column, and (2) were completed prior to creating dummy variable columns, which is described later.

### *3.3.5 Minimum Number of Rows*

After the removal of these columns and rows, one of the final filtering processes ensures that the dataset has a sufficient number of rows. In determining the necessary number of rows for the datasets, we were faced with several options. One option was to exclude datasets whose number of rows did not satisfy the requirements of the default settings of the selected metamodels. The most stringent algorithm, in terms of row quantity requirements per the default settings, was k-nearest neighbors (KNN). *Scikit-learn*'s implementation of KNN uses a default number of neighbors of $k = 5$. This necessitates at least five observations in each training dataset. Using the default training-testing split of 75% and 25%, respectively, seven observations is the smallest amount that permits five observations in the training dataset. There is the option to decrease the number of neighbors used in the algorithm to permit datasets with fewer than seven rows. However, the absolute minimum number of observations is three, in which two observations are used for training and one is used for testing. Two training points does not lend itself to much use, and so this option was quickly disregarded. Another option was to find a "happy medium" by adjusting both the minimum number of rows and the value of $k$ for KNN, but we found that keeping the default settings to be of greater value than permitting datasets with less than seven observations.

Datasets that do not satisfy the minimum number of rows requirement do not transition from the candidate problem space to the problem space, causing an additional 18 datasets to not make the cut. We also restrict datasets to a maximum of 10,000 observations,

as datasets with more rows than this threshold greatly increased computation time. This removed 15 datasets. After preprocessing, the minimum and maximum number of rows in the qualified datasets is 7 and 8,437, respectively.

### *3.3.6 Minimum Number of Numeric Columns – Post-binary Encoding*

To compute a gradient for the dataset, there must be at least two columns with numeric values. This check occurred after categorical data was transformed into a binary representation (and was, therefore, numeric), thus, increasing the likelihood of datasets meeting this requirement. There were 123 datasets that did not contain at least two columns with numeric values.

### *3.3.7 Removing Datasets with the Same Name*

The last filtering step checks if the name of the dataset already exists in the pooled data. Within the *pydataset* package, there are 60 instances in which a dataset has the same name as another one, and so only one instance of each dataset with the same name made it to the problem space. It was *not* checked that datasets with the same name in fact contained the same data.

### *3.3.8 Filtering Summary*

Datasets were removed from the problem space if they (1) did not have at least one numeric column prior to binary encoding (which removed 16 datasets); (2) did not meet the minimum number of rows requirement (which removed 18 datasets); (3) did not satisfy the requirement to have less than 10,000 rows (which removed 15 datasets); (4) did not meet the minimum number of numeric columns requirement after binary encoding (which removed 122 datasets); and (5) if the name of the dataset already existed in the problem

space (which removed 30 datasets). As previously mentioned, the candidate problem space consists of 758 datasets. After preprocessing, 557 datasets move onto the problem space.

### *3.3.9 Target Selection*

Each dataset must have a target column on which the various algorithms will make predictions. Options for selecting a target column include (1) hand-picking each target column, and (2) creating an algorithm to do so automatically. We chose to create an algorithm to select the target column, as this saves time and effort and because we may not have knowledge on the data. It should be noted that, when used in the real-world, the users of this framework should most likely already have target column(s) selected based on their dataset's purpose.

Datasets that are in the problem space are guaranteed to have at least one numeric column, as this was a requirement to transition from the candidate problem space to the problem space. The target variable selection algorithm sets the numeric column with the greatest number of unique values as the target column. For example, for the *Top Gear* dataset, the target column would be "Price", as this column contains the greatest number of unique values. However, because we are using the *Top Gear* dataset as an example throughout the paper, and because we have inspected the columns and their data, we set the target column as the one that made the most sense in context – the "Verdict" column, which contains the *Top Gear* television program's hosts' opinions of each car's overall value.

### *3.3.10 Datatype Classification*

As mentioned in Chapter II, there is a need to distinguish among different datatypes. This is prudent because we must ensure that variables are handled according to their

datatype, or, at least according to the datatype that they most resemble. Distinguishing between continuous, ordinal, and count data is not a task completed in Python. To distinguish between continuous and discrete data is a difficult process, as it is often greatly determined by context. The main preprocessing techniques that require knowledge of a variable's datatype are scaling and binary encoding.

The options moving forward in regards to scaling are to (1) leave all numeric data unscaled, (i.e., both continuous, count, and numeric discrete data), (2) scale all numeric data, or (3) attempt to distinguish between continuous, count, and numeric discrete data and only scale those that are regarded as continuous or count. The worst that could happen in option (3) is that discrete columns are scaled and/or continuous and count columns are not scaled, a fate that is certain of the other two options. Therefore, we decided to develop a proxy classification method, which is later described.

The choices moving forward regarding binary encoding are to (1) only encode variables determined by Python as categorical (i.e., only those that are textual), and (2) encode variables determined by Python as categorical, as well as numeric data that appears to be ordinal (rather than count). Option 2 requires automating a process to distinguish between count and ordinal variables. We decided to pursue option (2), which is now described.

We reasoned that the components of the datatype classification method must take into account the number of rows in the dataset, as well as the number of unique values in any given numeric column. The overall logic is that if the number of unique values in a column is close to the number of rows, this column should be considered continuous. The method is now detailed. Given a variable in a dataset, we first check if it is numeric. If it is

not, it is considered categorical. If a variable is numeric, we then check if it is Boolean, (i.e., if the only levels in the dataset are 0 and 1). If it is, we leave the variable as is, because binary encoding would not result in any changes. If the variable is not Boolean, we check the number of rows in the dataset. If the given dataset contains less than 30 rows, then the proportion of unique values in any given column to the number of rows must be less than or equal to 0.25 to be classified as a discrete variable. In this case, it is always categorized as ordinal (rather than possibly classifying it as a count). If this ratio is greater than 0.25, then it is considered continuous. If there are 30 or more rows, then we first check if there are 10 or fewer levels. If there are, we classify the variable as ordinal. If there are not, we look at the same ratio as previously described. If this ratio is less than or equal to 0.50, then we consider it count data. Otherwise, we consider it continuous. This process is captured in the flowchart in Figure 6, where blue diamonds represent decision points, yellow boxes represent that the variable is categorical, and orange boxes represent that the variable is *not* categorical.

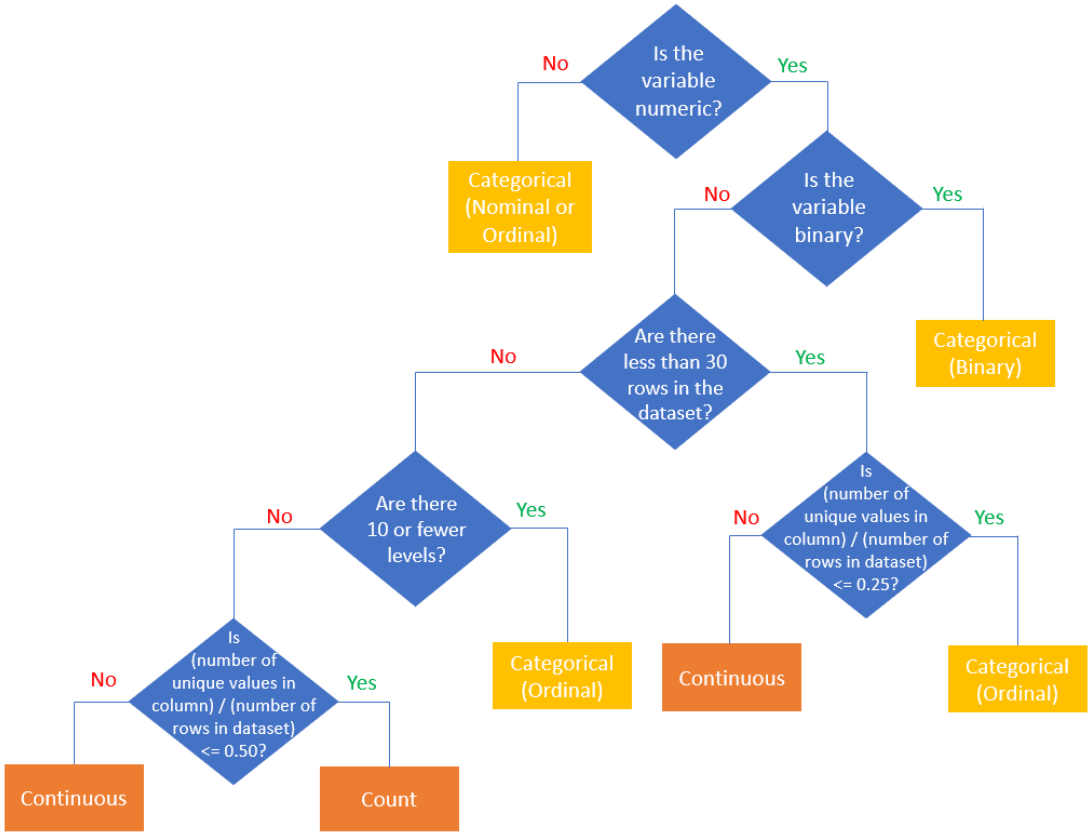


**Figure 6. Flowchart for determining datatypes**

As mentioned in the review of the literature, the datatype classification of different variables does not aim to accurately label variables according to their human-determined datatype, but rather according to the data itself. Nevertheless, it is beneficial to see how it performs according to what we would expect.

Table **4** presents the descriptions of each variable in the *Top Gear* dataset, along with their datatype classification according to the previously described algorithm. This dataset contains 238 cars (i.e., rows) and so the threshold used in the above algorithm was

0.25. The only variable whose classification could be challenged by human standards is miles per gallon.

**Table 4. Classification results for *Top Gear* dataset**

| Column | Description | Number of unique values | Proportion | Classification |
|---|---|---|---|---|
| Price | List price (UK pounds) | 228 | 0.957 | Continuous |
| Cylinders | Number of cylinders in the engine | 9 | 0.038 | Ordinal |
| Displacement | Displacement of the engine (in cc) | 97 | 0.408 | Continuous |
| BHP | Power of the engine (in bhp) | 120 | 0.504 | Continuous |
| Torque | Torque of the engine (in lb/ft) | 108 | 0.454 | Continuous |
| Acceleration | Time it takes car to get from 0 to 62mph (in seconds) | 98 | 0.412 | Continuous |
| TopSpeed | Car's top speed (in mph) | 82 | 0.345 | Continuous |
| MPG | Combined fuel consumption (urban + extra urban, in miles per gallon) | 56 | 0.235 | Count |
| Weight | Car's curb weight (in kg) | 195 | 0.819 | Continuous |
| Length | Car's length (in mm) | 204 | 0.857 | Continuous |
| Width | Car's width (in mm) | 143 | 0.601 | Continuous |
| Height | Car's height (in mm) | 172 | 0.723 | Continuous |

In reality, miles per gallon is a continuous variable, as it is a ratio measurement. No car has a perfect whole number MPG, but rather, there would always be some trailing decimals. But, as was mentioned regarding the length of a car in Chapter II, miles per gallon is rounded (and must be to be included in decimal form in a dataset). There are simply not enough unique miles per gallon values to classify this variable as continuous. Greater precision would result in a greater number of unique values, which would certainly change its classification to continuous, as it was just shy of doing so with the current precision. Miles per gallon inherently also has a low range, which also decreases the chance of its classification of continuous. Nevertheless, it presents itself as a discrete variable according to the algorithm and should thus be treated as one.

The algorithm does not classify target columns. However, we tested to see how it would label the *Top Gear* dataset's "Verdict" column (in which possible values are integers

from 1 to 10) and it correctly classified it as an ordinal variable. It should be noted that this is highly sensitive to the fact that 10 levels was used as the threshold for distinguishing between ordinal and continuous or count data. Because "Verdict" had exactly 10 levels in its data, it was considered ordinal. Had the algorithm's threshold been any smaller, or had a verdict of, say, 3 never appeared in the dataset it would have classified "Verdict" as a count variable.

Another dataset whose datatype classifications we inspected was the dataset *Baseball*, whose results are shown in Table 5. This dataset contains 263 observations, and so the ratio threshold is 0.25. The majority of the datasets of the variables are count variables describing the frequency of various measurements. However, most variables within this dataset are categorized as continuous, because of the variation in counts among the players for each statistic. Although it is understood that these variables are count variables, it makes more sense to treat them as continuous variables (i.e., scale these variables) to ensure that their magnitude does not affect their impact on the algorithms.

**Table 5. Classification results for *Baseball* dataset**

| Column | Description | Number of unique values | Proportion | Classification |
|---|---|---|---|---|
| atbat86 | Number of times at bat in 1986 | 209 | 0.795 | Continuous |
| hits86 | Number of hits in 1986 | 130 | 0.494 | Continuous |
| homer86 | Number of home runs in 1986 | 35 | 0.133 | Count |
| runs86 | Number of runs in 1986 | 92 | 0.350 | Continuous |
| rbi86 | Number of runs batted in 1986 | 94 | 0.357 | Continuous |
| walks86 | Number of walks in 1986 | 87 | 0.331 | Continuous |
| years | Number of years in the major leagues | 21 | 0.080 | Count |
| hits | Number of hits during his career | 241 | 0.916 | Continuous |
| homeruns | Number of homeruns during his career | 129 | 0.490 | Continuous |
| runs | Number of runs during his career | 226 | 0.859 | Continuous |
| rbi | Number of runs batted in during his career | 226 | 0.859 | Continuous |
| walks | Number of walks during his career | 207 | 0.787 | Continuous |
| outs86 | Number of put outs in 1986 | 199 | 0.757 | Continuous |
| assist86 | Number of assists in 1986 | 145 | 0.551 | Continuous |
| error86 | Number of errors in 1986 | 29 | 0.110 | Count |
| sal87 | 1986 annual salary on opening day in thousands of dollars | 150 | 0.570 | Continuous |

The following table lays out the preprocessing method we use for each datatype. Because we decided to treat both continuous and count data in the same manner, it does not matter if the algorithm correctly distinguishes between the two.

**Table 6. Preprocessing methods used per datatype**

| Datatype | Preprocessing Method |
|---|---|
| Continuous | Scale |
| Count | Scale |
| Categorical (only nominal and ordinal) | Binary encoding |
| Binary | None, as it is already "binary encoded" |

### *3.3.11 Scaling Method*

Because the *Scikit-learn* package does not use a default scaling method, we decided to use z-score standardization. One issue with using the min-max normalization is that it assumes that, in the real world, no values lie outside of the maximum and minimum ranges, an assumption that could cause troubles in testing. It was decided to standardize the data for *all* algorithms, rather than making this preprocessing technique metamodel-dependent.

### *3.3.12 Categorical Variables*

As mentioned in Chapter II, the literature describes methods for handling nominal versus ordinal variables. Distinguishing between the two types would require we inspect all categorical variables individually. Therefore, we handle all categorical variables as though they were nominal. This allows us to avoid treating a nominal variable as ordinal, which could have great repercussions.

For testing, we use one-hot encoding. Table 7 shows a side-by-side comparison of a subset of the original *Top Gear* dataset and a subset of the data with its corresponding dummy variables. The following illustration shows how the one-hot encoding works, using

the *Top Gear* dataset as an example. We take the "Origin" variable, which has three levels: "Asia", "Europe", and "USA". Only two columns are needed to represent data of three levels. Of the three levels, "Asia" comes first alphabetically, and so this level is not included as a column header. The original dataset on the left of Table 7 shows that Alfa Romeo is a European auto company. This is depicted in the table on the right via the "1" in the "Origin_Europe" column and a "0" in the "Origin_USA" column. Likewise, as shown in the left table, Ford is a U.S. auto company. This is represented in the table on the right by the "0" in the "Origin_Europe" column, coupled with the "1" in the "Origin_USA" column. As a last example, Subaru, which is an Asian company, receives "0"s for both "Origin_Europe" and "Origin_USA", indicating that it is of neither of these origins. Metamodels "do not care" that Subaru is an Asian company – they only care that it does not fall into one of the other two categories.

**Table 7. Comparison of original *Top Gear* dataset and dataset with dummy variables**

| **Original Dataset** | | |
|---|---|---|
| | **Maker** | **Origin** |
| 0 | Alfa Romeo | Europe |
| 2 | Aston Martin | Europe |
| 45 | Chevrolet | USA |
| 74 | Ford | USA |
| 189 | Subaru | Asia |
| 193 | Suzuki | Asia |

| **Dataset with Dummy Variables** | | | |
|---|---|---|---|
| | **Maker** | **Origin_Europe** | **Origin_USA** |
| 0 | Alfa Romeo | 1 | 0 |
| 2 | Aston Martin | 1 | 0 |
| 45 | Chevrolet | 0 | 1 |
| 75 | Ford | 0 | 1 |
| 189 | Subaru | 0 | 0 |
| 193 | Suzuki | 0 | 0 |

### 3.4 Meta-features

The meta-features are then calculated for each dataset. The purpose of the meta-features is to capture the inherent structure of the dataset, rather than looking at the dataset inputs directly. These meta-features can then be used concurrently to compare with the meta-features of a new dataset.

In accordance with Rice, meta-feature selection is a critical step in designing a recommendation system (Rice, 1976). Any variation in meta-features selected or in the number of meta-features could greatly impact the system's recommendation.

Simple meta-features were incorporated into the recommendation system, to include

- Number of observations (rows)
- Number of attributes (columns)
- Rows-to-columns ratio

In their paper, "Experimental Designs, Meta-modeling, and Meta-learning for Mixed-Factor Systems with Large Decision Spaces", Little et. al note that inclusion of meta-features capturing "statistical features of input types (categorical, discrete, and continuous) or number of levels" may improve the recommendation system (Little, 2018). To abide by this suggestion, the following meta-features are incorporated:

- Number of discrete columns
- Minimum number of factors among discrete columns
- Maximum number of factors among discrete columns
- Average number of factors among discrete columns
- Number of continuous columns

In their article, Cui et al. propose 15 meta-features for use in their recommendation system and an in-depth explanation of these meta-features can be found in their paper (Cui, Hu, Weir, & Wu, 2016). Of those suggested, along with the gradient minimum, the following are implemented:

- Gradient average

- Gradient maximum
- Gradient standard deviation

### 3.5 Training and Testing Sets

The data is then partitioned into a training and testing set. The rows are shuffled before the split to avoid any structural data collection method. The training set encompasses 75% of the data, with the remaining 25% residing in the testing set, which are the default percentages in *Scikit-learn*'s splitting method. For this research, the normalized root mean squared error (NRMSE) was utilized as the performance metric for each metamodel.

### 3.6 Metamodels

As previously mentioned, the Python package *Scikit-learn* was used to supply the metamodels. For the most part, those implemented used the default settings from *Scikit-learn*. The only deviation from the default settings is in the implementation of *k*-nearest neighbors. This algorithm defaults to uniform weights, but we use weights based on distance, as this aligned with the KNN method described in Chapter II. For other parameters and settings, please reference the *Scikit-learn* website (Pedregosa, 2011). The metamodels selected include:

- Bayesian ridge regression
- Decision tree regression
- Linear regression
- k-nearest neighbors regression
- Ridge regression

- Stochastic gradient descent

It should be noted that the selection of these particular models may not be "the ideal" subset. These are simply the ones placed in the algorithm space for testing.

### 3.7 Proposed Framework

The proposed framework is adopted from Rice's model and from Cui's model and is formed by two phases, as shown in Figure 7 (Rice, 1976; Cui, Wu, Hu, Weir, & Li, 2016).

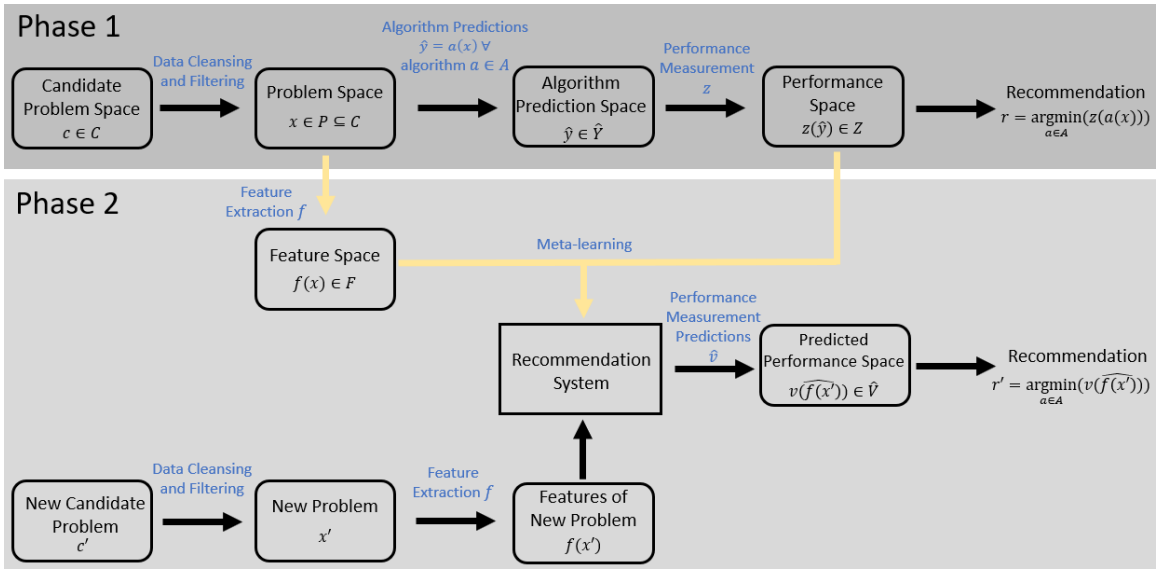


**Figure 7. Proposed framework for metamodel recommendation system**

The first phase is used to train and test datasets with the purpose of providing data for the recommendation system, and the second phase houses the recommendation system and allows for new datasets to be evaluated. Each phase is now described.

Phase 1 begins with the candidate problem space, $C$, which contains all potential datasets. Each dataset $c \in C$ is cleansed and is potentially deemed ineligible to enter the next stage in the phase. Those that move onto the next stage make up the problem space $P$,

where $x \in P \subseteq C$. Two steps occur from the problem space: feature extraction and algorithm predictions. Feature extraction, $f$, is applied to each $x \in P$ to give the feature space $F$, where $f(x) \in F$. This space is composed of the meta-features and is not used until Phase 2. The other step from the problem space is calculating the algorithm predictions. Each algorithm $a$ in the algorithm space $A$ is applied to each problem $x$, which then forms the algorithm prediction space $\hat{Y}$, where $\hat{y} = a(x) \in \hat{Y}$. A performance measurement $z$ is then applied to each prediction, leading to the performance space $Z$, where $z(\hat{y}) \in Z$. The recommended model, $r \in A$, is then selected by choosing the algorithm $a$ that minimizes $z(\hat{y})$, i.e., $r = \underset{a \in A}{\operatorname{argmin}}(z(a(x)))$.

The first part of Phase 2 uses the feature space $F$ and the performance space $Z$, as provided by Phase 1, along with meta-learning, to create the recommendation system. The framework learns what performance measurements for each algorithm can be expected, given specific data in the feature space. A new candidate problem, $c'$, enters the framework and is cleaned and filtered using the same cleansing and filtering steps of Phase 1. If this new candidate problem meets the preprocessing requirements, it becomes a new problem $x'$. Feature extraction $f$ is applied to the new problem to give the meta-features, $f(x')$. These features are put into the recommendation system which then provides performance measurement predictions $v(\widehat{f(x')}) \in \hat{V}$. The recommended metamodel is the one that minimizes the predicted performance measurement, i.e., $r' = \underset{a \in A}{\operatorname{argmin}}(v(\widehat{f(x')}))$.

The purpose of the framework is to enable the recommendation of a metamodel for a new dataset, without the need to run the dataset through each candidate metamodel, thus saving time and computation expenses.

### 3.8 Validation Run

The next step is to check the accuracy of the recommender. Every dataset runs through the entire system – that is, each dataset is individually preprocessed, given a target column, and meta-features are computed. In accordance with the defaults of *Scikit-learn*, we use a 75/25 percent split for the training and testing sets. The NRMSEs are dependent on the observations within the training and testing sets. For example, a model whose testing set contains an observation drastically different from any observation within the training dataset would receive a performance score worse than that of the model whose training set contained this observation rather than the testing set. Consequently, the NRMSEs are at the mercy of the training-testing split. To reduce the likelihood that a metamodel's potential poor performance is due to a poor training-testing split, we split the training and testing set 30 separate times. For each metamodel, we then decide which of these splits provided the best results (i.e., the lowest NRMSE) and take the model with that split as the representative model for that metamodel. This ensures that we pick a model that is better than (or at least as good as) 29 other models. A sample of this new dataset is shown in Table 8, where grey columns reference the dataset name, blue columns represent the meta-features (inputs), and green columns represent the NRMSEs (outputs).

**Table 8. Sample of meta-features and NRMSEs of *Top Gear* dataset**

| | | Meta-features | | | | | | NRMSEs | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| # | Dataset Name | Rows | Columns | Rows-Cols Ratio | ⋯ | Gradient Maximum | Gradient Standard Deviation | Bayesian Ridge | Decision Tree Regression | ⋯ | Stochastic Gradient Descent |
| 1 | top_gear | 238 | 30 | 7.933 | ⋯ | 43.546 | 3.538 | 0.14366 | 0.17970 | ⋯ | 1.515e+10 |
| 2 | HairEyeColor | 32 | 4 | 8 | ⋯ | 255.000 | 69.127 | 0.15706 | 0.12106 | ⋯ | 0.13276 |
| 3 | InsectSprays | 72 | 2 | 36 | | 255.000 | 71.195 | 0.13761 | 0.13490 | ⋯ | 0.24548 |
| 4 | LifeCylceSavings | 50 | 5 | 10 | | 5.561 | 1.061 | 0.10363 | 0.09772 | ⋯ | 0.64536 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 557 | sleepstudy | 180 | 3 | 60 | ⋯ | 1.000 | 109.465 | 0.16986 | 0.18638 | ⋯ | 6.53600e+10 |

The meta-learner in the recommendation system uses linear regression, where the inputs are the meta-features and outputs are the NRMSEs. To validate that the system works properly, we use a leave-one-out approach. The "training" dataset that builds the model is made up of $n - 1$ datasets, where $n$ is the total number of datasets in the system. Note that the term "dataset" in this case refers to the newly computed meta-features and the NRMSEs, rather than the inputs and output from the original datasets. The dataset that was *not* used to build the model is used for validation purposes. Taking the data in Table 9 as an example, the "training" dataset is comprised of observations 2 through 557. The first observation, *top_gear*, is used to validate the system. Once the model is built using the training dataset, we give the resulting model the meta-features of the *Top Gear* dataset and are returned with the predicted NRMSEs for each metamodel. Because we have already computed the actual NRMSEs for the *Top Gear* dataset, we can determine the performance of the metamodel by comparing the predicted an actual NRMSEs. This leave-one-out method is used for all 557 datasets.

**Table 9. Example of leave-one out method for *Top Gear* dataset**

| | | | Meta-features | | | | | | NRMSEs | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | # | Dataset Name | Rows | Columns | Rows-Cols Ratio | ⋯ | Gradient Maximum | Gradient Standard Deviation | Bayesian Ridge | Decision Tree Regression | ⋯ | Stochastic Gradient Descent |
| Testing → | 1 | top_gear | 238 | 30 | 7.933 | ⋯ | 43.546 | 3.538 | 0.14366 | 0.17970 | ⋯ | 1.515e+10 |
| Training | 2 | HairEyeColor | 32 | 4 | 8 | ⋯ | 255.000 | 69.127 | 0.15706 | 0.12106 | ⋯ | 0.13276 |
| | 3 | InsectSprays | 72 | 2 | 36 | | 255.000 | 71.195 | 0.13761 | 0.13490 | ⋯ | 0.24548 |
| | 4 | LifeCylceSavings | 50 | 5 | 10 | | 5.561 | 1.061 | 0.10363 | 0.09772 | ⋯ | 0.64536 |
| | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| | 557 | sleepstudy | 180 | 3 | 60 | ⋯ | 1.000 | 109.465 | 0.16986 | 0.18638 | ⋯ | 6.53600e+10 |

# IV. Analysis and Results

We calculate the accuracy of the recommendation system using several measurements. First, we look at the percentage of times that the system recommended the correct metamodel by counting the number of times the recommendation was the best metamodel and dividing it by the total number of recommendations. This gave us an accuracy of $127/557 \approx 0.228$. Although the best metamodel was recommended at an accuracy just above the likelihood of randomly guessing the correct metamodel (16.67% likelihood of guessing the best of six metamodels), this does not suggest that the system does not perform well. For example, several metamodels may perform well for any given dataset, and, therefore, selecting the second or third best option may be of satisfaction.

We also looked at the percentage in which the recommended metamodel's actual NRMSE was equivalent to the NRMSE of the actual best metamodel. This resulted in an accuracy of $130/557 \approx 0.233$. Comparing this with the accuracy presented in the previous paragraph, we observe that three recommended metamodels that were not the actual best metamodel had an identical NRMSE as the actual best metamodel.

Instead of only considering successful recommendations as those that were the actual best recommendation, we looked at whether the recommended metamodel's NRMSE lies within a certain epsilon of the best metamodel's NRMSE. We did this both in terms of whether the recommended metamodel's NRMSE falls within a certain percentage of the best metamodel's NRMSE, as well as in terms of the difference between the NRMSEs of the actual best and the recommended metamodel.

For the first method, we calculated the frequency in which the recommended metamodel's NRMSE was within 1%, 5%, and 10% of the best metamodel's NRMSE. The pseudocode for this method is shown below.

```
function(best_metamodel_NRMSE, recommended_metamodel_NRMSE, epsilon)
        # calculate lower bound
        lower_bound = best_metamodel_NRMSE*(1 + epsilon)
        # calculate upper bound
        upper_bound = best_metamodel_NRMSE*(1 – epsilon)
        # test
        IF recommended_metamodel_NRMSE >= lower_bound
                IF recommended_metamodel_NRMSE <= upper_bound THEN
                        OUTPUT "This recommendation is good."
                OTHERWISE
                        OUTPUT "This recommendation is bad."
        OTHERWISE
                OUTPUT "This recommendation is bad."
```

The percentage of recommended metamodel's whose actual NRMSE fell within an epsilon of 1% of the best metamodel's NRMSE was about 28.9%. Epsilons of 5% and 10% gave us results of about 34.5% and 38.1%, respectively. These results are displayed in Table 10.

**Table 10. Percentage of recommendations whose NRMSE fall within an epsilon percentage of the best metamodel's NRMSE**

| Epsilon | Percentage of Recommendations |
|---|---|
| 0.01 | 28.9% |
| 0.05 | 34.5% |
| 0.10 | 38.1% |

For the difference between the best and recommended metamodel's NRMSE, we used epsilon values of 0.01, 0.05, and 0.50. Pseudocode for modeling this is shown below.

```
function(best_metamodel_NRMSE, recommended_metamodel_NRMSE, epsilon)
        # calculate lower bound
        lower_bound = best_metamodel_NRMSE – epsilon
        # calculate upper bound
        upper_bound = best_metamodel_NRMSE + epsilon
        # test
        IF recommended_metamodel_NRMSE >= lower_bound
                IF recommended_metamodel_NRMSE <= upper_bound THEN
                        OUTPUT "This recommendation is good."
                OTHERWISE
                        OUTPUT "This recommendation is bad."
        OTHERWISE
                OUTPUT "This recommendation is bad."
```

The results from these accuracy measurements are promising. The percentage of recommended metamodels whose actual NRMSE fell within an epsilon of 0.01 of the best metamodel's NRMSE is about 40.6%. Epsilon values of 0.05 and 0.10 gave us results of about 70.6% and 90.1%, respectively. These results are shown in Table 11.

**Table 11. Percentage of recommendations whose NRMSE fall within a decimal valued epsilon of the best metamodel's NRMSE**

| Epsilon | Percentage of Recommendations |
|---|---|
| 0.01 | 40.6% |
| 0.05 | 70.6% |
| 0.10 | 90.1% |

Of the two measurements presented above, we argue that the more informative option is to look at the difference between the best and recommended metamodel's NRMSE. Requiring the recommended metamodel's NRMSE to fall within a specific constant value of the best metamodel's NRMSE creates a level playing field for all NRMSEs, as the results are only dependent upon the predefined epsilon. On the other hand, looking at whether the recommended metamodel's NRMSE falls within a certain

percentage of the best metamodel's NRMSE depends on the percentage that we choose, as well as on the best metamodel's NRMSE. To elaborate, as the best metamodel's NRMSE decrease (i.e., gets better), the window of accepting the recommended model as a good prediction decreases as well. We argue that the potential good performance of the best metamodel should not endanger the recommended model from being considered "good", nor should the potential poor performance of the best metamodel improve the likelihood that the recommended metamodel is labeled as a "good" recommendation.

Another approach to evaluate the recommendation system is to consider how the recommended best metamodel performs in comparison to the actual best metamodel. This is another way to see if the recommendation system gives "good" predictions, in lieu of the "best" prediction. This can be calculated via a relative performance score, where higher scores are better, with a score of 1 indicating that the recommended metamodel and the best metamodel had identical NRMSEs. It should be noted that if the predicted best metamodel's actual NRMSE score is 0, it is known that the actual best metamodel's NRMSE score is 0 as well. This scenario was assigned a relative performance score of 1, to avoid dividing by 0. After the runs, the average relative performance was 0.763, meaning that, on average, the predicted metamodel was 76.3% as good as the best metamodel. However, because the average and medians of this distribution are not the same (with an average of 76.3% and a median of 81.7%), 50% of the data does not lie above or below the average. About 29% of the recommendations have a relative performance score between 96% and 100%, as shown in Figure 8.

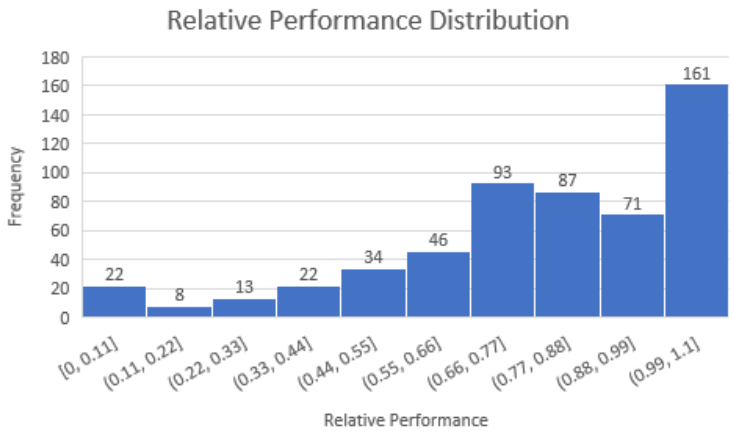


**Figure 8. Distribution of relative performances**

One downside to the relative performance approach is that an NRMSE of exactly 0 for the actual best metamodel will give a relative performance score of 0, unless the predicted metamodel has an NRMSE of 0, as previously described. This result could possibly not accurately reflect the predicted metamodel's performance compared to the best. For example, one would claim that a predicted NRMSE of 0.005 is very close to an actual NRMSE of 0. However, the relative performance score would show otherwise.

Of the 557 predictions, there were twelve instances in which the best metamodel's NRMSE was 0. It should be noted that the data providing a perfect NRMSE were inspected after the model was run. The target column selected for these datasets was either (1) a column identical to an input column, (2) a linear combination of a portion of the other columns, (3) a combination of other columns and a constant, such as 100, or (4) an instance in which values from one column directly determined the output. For the fourth reason listed, an example is the "iraqVote" dataset. The dataset contains 100 rows. One column contains one of 50 states, where each state was represented in two rows. The target column, "gorevote" is "the vote share recorded by Al Gore in the corresponding state in the 2000

Presidential election" (Arel-Bundock, 2019). This signifies that each state has its own value, and so the value of the state directly led to the value of the target column.

Of the datasets that resulted in a perfect NRMSE, none of the predicted best metamodel's NRMSE was equal to 0, meaning that none of the twelve predictions for these datasets received a good relative performance score. Of the seven instances, four of the recommended metamodel's NRMSE was less than 0.05 (i.e., very close to the actual best metamodel's NRMSE). Although these NRMSEs of the predicted best metamodel are close to the NRMSEs of the actual best metamodel, the relative performance score shows otherwise. With this in mind, it can be concluded that the metamodel recommendation system gives as good of a prediction as the best metamodel *more than* 76.3% of the time, on average. Excluding these twelve instances from the relative performance calculation gives a new average relative performance score of $424.76/(557-12) \approx 0.779$, where 424.76 is the sum of the relative performance scores excluding the seven instances described above.

# V. Conclusions and Recommendations

This research aims to expand upon the work offered by Cui et. al by creating a metamodel recommendation system that works with data of all shapes and sizes. The framework mimics that of Cui's and also allows for a large quantity of datasets, as well as implements data preparation.

As the results show, the metamodel recommendation system provides good recommendations when given an "unknown" dataset. Although it does not tend to suggest the best metamodel, it more often than not provides a recommendation that is very comparable to the best metamodel.

Because the metamodel recommendation system makes good recommendations, it can be used to make recommendations for datasets without previously computing that dataset's actual NRMSEs. Following the framework proposed in Section 3.7, a new dataset is preprocessed and, if it meets the requirements to enter the problem space, its meta-features are calculated. After this, its NRMSE for each metamodel are predicted using the constructed meta-learner.

As with any study, ours can be expanded upon and improved. Each component of the metamodel recommendation system could benefit from additional components and could be strengthened through additional research. We propose several areas for future research in the following paragraphs.

The following recommendations refer to the datasets used, as well as data preparation.

- The metamodel recommendation system only uses one Python library from which to pull datasets. Additional libraries exist which could allow for more datasets, possibly increasing the variety and, thus, improving the likelihood that a newly

introduced dataset would have a similar counterpart that was already put through the system.

- The framework does not check the similarity among datasets. The decision to maintain datasets with similar meta-features would need to be made. Additionally, other than the check to ensure that datasets with the same name do not all make it to the problem space, we do not check that each dataset is unique. For example, by chance, we found that three datasets, each with a different name, all contained the exact same data: Bechtoldt, Bechtoldt.1, and Bechtold.2. Because each of these are handled separately, they all have different training-testing splits which could result in different metamodels recommendations. This could pose a problem when the system predicts the best metamodel for another dataset with similar (or identical) inputs.
- Section 2.4.7 provides additional data preparation steps that could be implemented into the system, such as outlier detection, determining the importance of each field, and feature-reduction techniques. However, as has been stressed throughout, steps such as these often require knowledge of the datasets.

The realm of meta-features lends itself to additional research and consideration. The following questions could be explored:

- What is the appropriate set of meta-features that reveal enough information about the datasets?
- Should the set of meta-features be data-dependent? That is, should use of a meta-feature depend on the data itself?

Although it was made clear that the recommendation system does not use the "ideal" set of metamodels, we were required to select several for implementation. Additional research could be done regarding the pool of metamodels and could aim to answer the following question:

- What is an appropriate set of "unique" metamodels? That is, regression metamodels, for example, could tend to perform similar. With that in mind, what is the appropriate set that includes a wide variety of metamodels, such as regression-based, neural networks, and decision trees?

Research could be done regarding meta-learners. The system implemented a linear regression learner, but that is not to say that this is the ideal meta-learner. We recommend addressing the following questions in future research:

- Does the meta-learner depend on the datasets? If so, what type of meta-learner should be used for specific datasets?
- Can a pool of meta-learners be implemented into the system?

Additional performance measurements could be selected, such as the following:

- Spearman's rank correlation to determine how well the system ranks the metamodels from best to worst
- A calculation of the frequency of which the system predicts the second best metamodel

- A calculation of the frequency of which the system predicts the worst metamodel

# Appendix A

For a copy of the code, please contact Dr. Weir at Jeffery.Weir@afit.edu. The code was developed and run on a 64-bit Intel® Core™ i5-8250U CPU at 1.60GHz. We use Python version 3.6, along with the following modules.

**Table 12. Python modules**

| Module | Version | Installation |
|---|---|---|
| numpy | 1.13.3+mkl | Anaconda package |
| operator | | Base package |
| pandas | 0.21.0 | Anaconda package |
| pydataset | 0.2.0 | pip install |
| scipy | 1.0.0 | Anaconda package |
| sklearn | 0.19.1 | pip install |
| statistics | | Base package |

# Bibliography


Acharya, M. S., Armaan, A., & S, A. A. (2019). A Comparison of Regression Models for Prediction of Graduate Admissions. *Second International Conference on Computational Intelligence in Data Science* (pp. 32-33). Chennai, India, India: IEEE.

Arel-Bundock, V. (2019, April 17). *iraqVote*. Retrieved February 15, 2020, from GitHub: https://vincentarelbundock.github.io/Rdatasets/doc/pscl/iraqVote.html

*Baseball Data*. (n.d.). Retrieved February 13, 2020, from https://vincentarelbundock.github.io/Rdatasets/doc/ISLR/Hitters.html

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning.* New York: Springer.

Cui, C., Hu, M., Weir, J. D., & Wu, T. (2016). A recommendation system for meta-modeling: A meta-learning based approach. *Expert Systems with Applications*, 33-44.

Cui, C., Wu, T., Hu, M., Weir, J. D., & Li, X. (2016). Short-term building energy model recommendation system: A meta-learning approach. *Applied Energy*, 251-263.

Ette, E. I., & Williams, P. J. (2007). *Pharmacometrics: The Science of Quantitative Pharmacology.* Hoboken, New Jersey: John Wiley & Sons, Inc.

Hoerl, A. E., & Kennard, R. W. (1970). Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics*, 55-67.

Larose, D. T., & Larose, C. D. (2014). *Discovering Knowledge in Data: An Introducion to Data Mining* (2nd ed.). Hoboken, New Jersey: John Wiley & Sons, Inc.

Little, Z. C. (2018). *Experimental Designs, Meta-modeling, and Meta-learning for Mixed Factor Systems with Large Decision Spaces.* Wright Patterson Air Force Base: Air Force Institute of Technology.

Narsky, I., & Porter, F. C. (2014). *Statistical Analysis Techniques in Particle Physics: Fits, Density Estimation and Supervised Learning.* Weinheim, Germany: Wiley-VCH.

Pedregosa, F. (2011). *Scikit-learn: Machine Learning in Python*. Retrieved from Scikit-learn.

Piccini, N. (2019, July 16). 101 Machine Learning Algorithms for Data Science with Cheat Sheets.

Rice, J. R. (1976). The algorithm selection problem. *Advances in Computers*, 65-118.

Savchenko, Y., & Stepashko, V. (2018). Metamodeling as a Way to Universalization of Inductive Modeling Tools. *2018 IEEE 13th International Scientific and Technical Conference on Computer Sciences and Information Technologies* (pp. 444-447). Lviv, Ukraine: IEEE.

*TopGear*. (n.d.). Retrieved 2 13, 2020, from R Documentation: https://www.rdocumentation.org/packages/robustHD/versions/0.5.1/topics/TopGear

Valera, I., & Ghahramani, Z. (2017). Automatic Discovery of the Statistical Types of Variables in a Dataset. *International Conference on Machine Learning. 70*, pp. 3521-3529. Sydney: Proceedings of Machine Learning Research.

Xu, W. (2011). Towards Optimal One Pass Large Scale Learning with Averaged Stochastic Gradient Descent. *Semantic Scholar*, arXiv preprint.

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 074-0188*



| **1. REPORT DATE** *(DD-MM-YYYY)* | **2. REPORT TYPE** | **3. DATES COVERED** *(From – To)* |
|---|---|---|
| 3-26-2020 | Master's Thesis | June 2017 – March 2020 |

**TITLE AND SUBTITLE**

A Metamodel Recommendation System using Meta-learning

**5a. CONTRACT NUMBER**

**5b. GRANT NUMBER**

**5c. PROGRAM ELEMENT NUMBER**

**6. AUTHOR(S)**

Woods, Megan K., Ctr

**5d. PROJECT NUMBER**

**5e. TASK NUMBER**

**5f. WORK UNIT NUMBER**

**7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)**
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/ENY)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

**8. PERFORMING ORGANIZATION REPORT NUMBER**

AFIT-ENS-MS-20-M-182

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
Intentionally Left Blank.

**10. SPONSOR/MONITOR'S ACRONYM(S)**

**AFRL/RHIQ (example)**

**11. SPONSOR/MONITOR'S REPORT NUMBER(S)**

**12. DISTRIBUTION/AVAILABILITY STATEMENT**
**DISTRUBTION STATEMENT A.** APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

**13. SUPPLEMENTARY NOTES**
This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

**14. ABSTRACT**

The importance and value of statistical predictions increase as data grows in availability and quantity. Metamodels, or surrogate models, provide the ability to rapidly approximate and predict information. However, selection of the appropriate metamodel for a given dataset is often a task, and the choice of the wrong metamodel could lead to considerably inaccurate results. This research proposes and tests the framework for a metamodel recommendation system. The implementation allows for virtually any dataset and preprocesses data, calculates meta-features, evaluates the performance of various metamodels, and learns how the data behaves via meta-learning, thus preparing and bettering itself for future recommendations. Testing on over 500 widely varied datasets, the framework provides positive results, often recommending a metamodel with similar performance as the actual best metamodel.

**15. SUBJECT TERMS**
Recommendation system, Metamodels, Meta-learning, Data Preparation

| **16. SECURITY CLASSIFICATION OF:** | | | **17. LIMITATION OF ABSTRACT** | **18. NUMBER OF PAGES** | **19a. NAME OF RESPONSIBLE PERSON** |
|---|---|---|---|---|---|
| **a. REPORT** | **b. ABSTRACT** | **c. THIS PAGE** | | | Dr. Jeffery Weir, AFIT/ENS |
| **U** | **U** | **U** | **UU** | 69 | **19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-3636, ext 4523 (Jeffery.weir@afit.edu) |

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