Introducing Stata to Super Learning

by

Thomas Carpenito

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Arts

in

Biostatistics

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Alan E. Hubbard, Chair
Professor Mark J. van der Laan
Professor Sophia Rabe-Hesketh

Spring 2018

Introducing Stata to Super Learning

Abstract

Introducing Stata to Super Learning

by

Thomas Carpenito

Master of Arts in Biostatistics

University of California, Berkeley

Professor Alan E. Hubbard, Chair

The current state of machine learning and modern statistical methods in the programming software Stata is almost non-existent. Should users require the use of machine learning techniques they must: program and develop a package themselves, rely on the Stata community to program a function to fit their needs, wait for Statacorp to release a new series of functions, or simply use another programming language. There are many potential reasons why such methods do not exist in Stata however one should not consider a piece of statistical software complete unless it provides an entirety of functionality to all audiences. Considering the large and diverse user base of Stata, there is the opportunity to reach a wide audience and "normalize" machine learning while satisfying a definite want and need in the community. In an attempt to modernize Stata, this thesis both contributes and documents a new data adaptive method (super learner) for the statistical software program. In essence, the super learner algorithm takes a series of candidate algorithms and optimally combines them into one. This introduced function allows the user to identify a discrete super learner winner, create an optimal combination of algorithm weights, make predictions on an out of sample dataset, and self evaluate through cross validation. Due to its complex functionality and customizability, the `superlearner` program in Stata is not fully complete; as a result, it has been made open source in the hopes that others may help continue to enhance the current code base. While this package does not fix the lack of machine learning in Stata, it does provide an additional tool for the statistician's toolkit and further supports the need for modern statistical methods in Stata.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Considering the relative history of the field of statistics, the use of machine learning and data adaptive methods is not particularly new; for example, while the term "logistic regression" was first coined in the 1940s, the first instances of classification and regression trees arrived only thirty years later with contributions by Breiman, Friedman, Olshen and Stone [10]. Yet, even though the work put forth by these statisticians nearly four decades ago is certainly well established, it is still met with unwavering criticism and hesitation [1]. These hindrances have undoubtedly made a lasting impact on how we apply theory to real life situations. For example, in the health care industry, many are hesitant to entrust a machine with a task for either safety concerns or for monetary reasons [8]. When dealing with human safety, caution is to be expected. However, one could argue by avoiding modern statistical methods we are simply turning a blind eye to recent advances in the field, and unknowingly providing a disservice to society. These hesitations and preferences may seem benign at first however the stigma attached to machine learning algorithms has had tremendous effects on the technology we use today. The use of new technology can be intimidating and daunting, yet one must remember that data adaptive estimation fundamentally builds off of much of classical statistics [10].

One of the more common arguments against modern statistical methods is that they are not completely well understood and can only be seen as "black box" models; individuals care only about what information goes in, what comes out, and not necessarily what happens in between. The paper titled "Statistical Modeling: The Two Cultures" drew attention to the divide between the data modeling and algorithm modeling worlds and encouraged individuals to see both perspectives and understand that statisticians must come together and collectively move forward to adopt an evolving set of tools [1]. Indeed, there is a great loss of interpretation when using machine learning algorithms when compared to data modeling techniques however this reasoning alone should not be responsible for discarding an entire subset of statistical models.

An alternative reason for not using machine learning algorithms is the belief that the software that oftentimes accompanies these models are untested and unstable. While untrue [24], this argument is likely related to the differences between proprietary and open source

software, not necessarily the divide amongst modern and classical statistical methodology. This argument may seem abstract and unrelated, however it is pertinent to the discussion of modern statistical methods particularly in regards to the programming software, Stata.

Regardless of one's own beliefs, the responsibility for analyzing any sort of data should fall to the statistician, not the technology. Modern statistical software provides the toolbox for the expert and such knowledge should be applied skillfully and thoughtfully. If the tools are not provided then the work can not be completed. As an example, the status for machine learning techniques in the statistical software package Stata is barren; considering Stata was first developed in 1985 [19], the fact that there is not a single natively implemented machine learning package built into the software is quite puzzling. There is a definite need for machine learning in Stata and that need is not currently being met.

In an attempt to remedy this issue, the completion of this thesis will add a single machine learning algorithm (super learner [17]) to the Stata software while encouraging the development of more modern statistical methods. Given the high customization and functionality of the super learner algorithm, this project will be ongoing and continuously improved as more individuals contribute other machine learning algorithms to the Stata software.

In the following sections this thesis will provide a synopsis of the current state of machine learning techniques in Stata and call attention to why these methods are needed. This thesis will then provide a gentle introduction to super learning and motivate why such a piece of software is needed. The middle section of the paper will focus on the logistics and syntax of creating the Stata package with examples on both simulated and actual data. Finally, a discussion is warranted on how the project will continue and what steps are necessary to encourage further data adaptive approaches to estimation in Stata.

Chapter 2

Current Status Of Machine Learning in Stata

If one were to launch Stata and search the GUI (graphical user interface) for a machine learning algorithm, they would be sorely disappointed to not find any. Stata provides a myriad of statistical models that can handle survival analysis, baeysian methodology, time series data, and even non parametric analyses however (as of this writing) not a single machine learning algorithm has been natively implemented in Stata by Statacorp. As previously discussed, regardless if one "believes" in the use of machine learning algorithms, the support to carry out an analysis with a desired statistical model should be implemented.

It is important to note that by providing machine learning algorithms, Stata is not actively suggesting one model (or school of models) is better than another. This is because Stata does not absolve the individual from blindly using any function without careful consideration nor does it provide a safety net for assumption free statistics. For example, many believe when analyzing survival data, the use of Cox-Proportional hazards is quite careless especially if the proportional hazards assumption has been violated [13]. One can easily load such a dataset into Stata and run an analysis without the software ever knowing about such a violation. This is in part because the responsibility of proper model formulation falls on the skilled statistician and also because Stata's main objectives is to "put hundreds of statistical tools at [the user's] fingertips" [19]. In order to fulfill this desire, Stata should consider the implementation of more modern statistical methodologies. One must remember that it is not Stata carrying out the analysis, but rather the Statistician.

However, this is not to say that machine learning does not exist in Stata. In fact, there are a few packages developed by the community for the use of modern statisical methodology. Some examples of such software include cross validation (`crossvalidate` [7]), lasso and elastic net regression (`lassoregress` [26]), ChaidForest for interaction detection(`chaidforest` [20]), and Support Vector Machines (`svmachines` [11]). While there is some progress being made in Stata, the number of machine learning algorithm pails in comparison to what can be easily found in other programming languages such as python [9] and R [21]. This then begs the question, why does Stata need machine learning techniques when they are already

readily accessible in other languages?

2.1 Why Stata?

According to the Stata website, there are over one hundred different fields of study ranging from econometrics to pharmaceutical research all of whose statistical needs are met with Stata software [19]. With such a broad range of functionality, one can assume the volume of daily active users must be quite large. This large user base can be at least partially explained by one of the most attractive features of Stata - its easy to use point and click GUI for assembling models. For example, if one wants to write a linear regression model, one simply clicks through a series of prompts and is able to build a model instantaneously. The threshold for using Stata becomes quite low; while there are certainly difficulties to programming Stata, one does not need to be a programmer to run any sort of analysis (this is contrasted to other languages such as python and R, for example). As one will see later, the use of an interactive GUI matches perfectly with the high customization of the super learner algorithm. Simply put, Stata software reaches a variety of people, across many different fields, all of which have different needs. Given the Stata user base and accessibility of the software, there is undoubtedly much need for the implementation of modern statistical software and also a great opportunity to introduce users to modern statistical methodology.

In terms of deferring users to another platform for machine learning, one must consider one of the largest pitfalls of open source software (OSS); anyone who can write a package is allowed to do so and publish with little to no accountability. This is not to say that all OSS is nonsense, nor should it be inferred that OSS is worse than proprietary software. With OSS there is far more room for creativity which leads to (as previously defined) a larger variety of tools at the statisticians disposal [24]. By contrast, many use Stata because it can be trusted, as each algorithm is well tested, only surfacing in Stata software after it has been internally certified [19]. This ensures there are little to no snags in analyses however this severely limits what the user can and cannot do when presented with a dataset. Ideally, one would like the best of both worlds and have a large number of properly validated, documented, and trusted tools at the user's disposal. The push for machine learning algorithms in Stata is supported by the fact that the infrastructure for properly documenting, testing, and validating statistical models already exists, it just has not yet been implemented for machine learning algorithms.

2.2 Project Goals

The first goal of this thesis is to provide a single machine learning algorithm (super learner) to the Stata library. As previously mentioned, the number of data adaptive algorithms that currently exist in Stata is almost non existent. While this project only adds a single tool to

the statistical toolbox, one will quickly see how super learning begs the addition of further algorithms and encourages future projects.

Secondly, this project hopes to stir interest in machine learning and provide a low barrier of entry for those individuals who are either intimated by modern statistical software or who have not heard of ensemble learning. The wide variety of individuals who use Stata is quite expansive; the opportunity to implement a piece of software to be used by individuals around the world is an opportunity to both modernize and normalize machine learning for future analyses.

Finally, this project hopes to garner enough traction and attention to provide evidence to Stata Corp that the need for modern statistical software is not only wanted but overdue. There is little reason as to why Stata should not adopt and implement modern statistical software, especially given their mission to "put hundreds of statistical tools at [the user's] fingertips." While users can submit their own Stata software packages to be downloaded, much of the usability, stability, and reliability in Stata functions are derived from the facet of being a proprietary software.

Chapter 3

Super Learning

The intent of this section is to briefly provide the reader with an overview of what super learning is. While the full theory behind super learner can be found in multiple publications [18] [17], this topic is outside the scope of the current thesis. As such, we direct the reader to the previously cited articles should they want to learn more about the properties of this machine learning algorithm.

In general, it can be quite difficult to properly model the relationship between an outcome and set of predictors with a single model, especially if that model is misspecified and parametric [18]. For example, if one was interested in modeling the relationship between coronary heart disease and an individual's: smoking status, weight, activity level, age, and diet, how would one select not only the best but most appropriate model? Is logistic regression relevant? If so, which covariates do we consider and is there an interaction term? Should one use a decision tree? Would neural networks perform better than support vector machines in this instance? How can one properly assert a model to describe the data before looking at the data? It is true, one could simply use an error metric to measure the difference between predicted and observed values and select the model with the lowest error rate [10], however this will only provide the best local algorithm. What if in reality the true underlying data generating system came from a combination of the previously mentioned models? The goal of super learning is simple: find the optimal combination of a number of candidate estimators (for example: logistic regression, random forest, neural networks, and support vector machine models) to best predict an outcome given a set of covariates [18].

3.1 Algorithm Overview

Candidate Library

Arguably, the most important part of the super learner algorithm is selecting the appropriate candidate library of estimators. The library is defined as a list of all possible algorithms the user believes may be either fully or partially responsible for modeling the relationship

between outcome and predictor variables. There is essentially no limit on what models a user may specify in their library, nor is their a limit on the number of candidate algorithms one may specify. Ideally, the algorithms in the candidate library should vary from one another however this is not a requirement of the super learning theory. In fact, there is only a benefit to adding more algorithms to the library [18]. The only restriction is that a user must (or should) specify which candidate algorithms they would like to include *a priori* or before looking at the data [17]. This ensures the user is not simply selecting a model based solely on the current data set and not the underlying data generating mechanism (commonly known as "data dredging" [2]). Of course, the methodology of selecting a model before looking at the data is not unique to the super learner algorithm itself and should be applied even when carrying out a simple regression.

Cross Validation

Generally speaking, once the candidate library has been selected the algorithm may begin "super learning". First, the original dataset is partitioned into K number of folds where $\frac{N}{K}$ observations fall into each fold (where N represents the total number of observations in the dataset). Next, one of the folds is held aside (known as the validation set) while the remaining $K-1$ folds (training set) are used to fit each successive model (separately) specified in the candidate library. Each proposed model is fit on the validation set and predictions are gathered. An error function is applied (typically the mean squared error), assessing the differences between our model prediction and the actual data outcome. This process is repeated until each fold has had an opportunity to be considered a validation set. The results of the loss function are averaged over all K iterations and an average loss for the particular model is reported. This process, more commonly known as "K-fold cross validation" is quite customizable, and more information about the various changes to the algorithm and general theory can be found elsewhere [10] [15] [18]. At this point, the candidate algorithm with the smallest cross validated risk is defined as the "discrete super learner winner."

Optimization of Weights

Once the discrete super learner has completed, the algorithm then proposes a family of weighted combinations (one for each algorithm in the user defined library) which minimizes the cross validated risk over the family of weighted combinations [18]. In essence, the algorithm both asks and answers the question: what combination of weights for each of my algorithms will predict my outcome with the lowest risk estimate? Further theory about how the optimization of weights is computed can be found in the publishable super learner paper [17] however it should be noted that the process of optimization is quite flexible and malleable [10] [16].

For this project, the optimization technique used was based off the idea of constructing a linear regression with constrained proportional coefficients whose sum adds to one [4].

Specifically, if our candidate library consisted of three algorithms, we would expect our linear equation to look something like equation 3.1.

$$y = a_1x_1 + a_2x_2 + a_3x_3 + a_4 + \epsilon \tag{3.1}$$

First, one can begin by constructing a series of generic equations (3.2, 3.3, 3.4) where only two of the three coefficients need to be estimated.

$$a_2 = \frac{e^{t_2}}{(1 + e^{t_2} + e^{t_3})} \tag{3.2}$$

$$a_3 = \frac{e^{t_3}}{(1 + e^{t_2} + e^{t_3})} \tag{3.3}$$

$$a_1 = \frac{1}{(1 + e^{t_2} + e^{t_3})} \tag{3.4}$$

Next, an arbitrary nonlinear function may be fit by the method of least squares using each candidate algorithm along with its corresponding generic equation [3]. This would be sufficient if we were interested in generating optimal weights for our regression coefficients of estimators without any constraints, however one must calculate a non linear combination of these estimators using the delta method to ensure all coefficients sum to one and are greater than zero [25] .

Predictions and Further Cross Validation

After the optimal weights have been calculated, each of the candidate estimators is run on the entirety of the dataset, the proper weights of each algorithm are applied, and the final super learner predictions are gathered. This new ensemble model may then be used to make predictions on a test set where the outcome is unknown. If an individual is interested in the performance of the super learner, one may perform cross validation on the super learner itself and compare the super learner algorithm against each of the candidate estimators specified in user defined library.

3.2 Further Notes

While the previous section has outlined the basics of the super learner algorithm, there is the option to fully customize almost each and every step of the process. Obviously one is able to select any number of candidate estimators deemed necessary for answering the question at hand (whether these algorithms be parametric, non-parametric, or semi-parametric) however, the algorithm further supports the customization from the number of folds used in validation, to the error family distribution, method for estimating weights, cluster identification, and if observational weights need be applied [23].

One can quickly see the benefit of super learning and understand its predictive power when trying to identify a relationship between outcome and set of predictors. However, one can also understand how the super learner methodology could be easily abused, having an individual simply select every possible algorithm into the candidate library and letting the machine learning take control of the situation. While this is a valid approach to using super learner, it is quite careless and may be costly. Care must be taken when specifying a candidate library as one will quickly learn the amount of computing time greatly increases as the number of candidate estimators grows. To summarize, the super learner algorithm does not allow for careless or assumption free statistics; the nature of ensemble learning provides as good as (if not better) results as simply picking a single model alone [18] however some thought must be given to what candidate library of algorithms one selects.

Chapter 4

`SuperLearner` in Stata

The super learner package in Stata (hereafter referred to as `superlearner`) has been built to accomplish four main tasks of the original super learner package: first, the package is able to select a discrete super learner winner, second, it formulates the optimal model, third, it will perform a cross validation on the proposed super learner, and finally, make out of sample predictions on a novel data set. This section describes the general syntax and flow of the `superlearner` package in Stata. While this section talks about the Stata code itself, we direct the reader to the discussion for future directions, limitations, and difficulties programming the `superlearner` algorithm.

4.1 Syntax

superlearner *depvar predictors*, [**k(**#**) library(***string***) superpredname(***string***) superestname(***string***) newdata(***file***) originaldataset(***file***) libraryglobals(***file***) family(***string***) loud evalmetric(***string***)**]

Settings

- **depvar**: A single dependent variable (also know as the response or outcome variable).
- **predictors**: A space separated string of predictors for which are used in your model(s).
- **k(**#**)**: The number of folds desired for the K fold cross validation. By default this package selects ten folds. The user supplied number of folds cannot be less than two nor can it be greater than 100.
- **library(***string***)**: A space separated string of both built-in and custom models to be included in the super learner algorithm. More information following.
- **superpredname(***string***)**: A user supplied string for what to name the predictions from the super learner algorithm.

- **superestname(***string***)**: A user supplied string for what to name the preserved super learner model to allow for out of sample predictions.
- **newdata(***file***)**: A valid data file for which the out of sample predictions will be generated on.
- **originaldataset(***file***)**: The name of the original dataset in which the super learner model was built.
- **libraryglobals(***file***)**: A valid .ado file which lists the custom user defined models.
- **family(***string***)**: The family error distribution. Accepts only the values of "Gaussian" or "Binomial".
- **loud**: If specified, the output from all commands (estimations, model fits, etc...) will be displayed. By default, these messages are suppressed.
- **evalmetric(***string***)**: A single string specifying the metric to be used for cross validation. Currently the supported error metrics are: mean square error (MSE), root mean square error (RMSE), mean absolute error (MAE), pseudo r^2 (r2), and area under the curve (AUC).

4.2 Using the Package

The general control flow for using the `superlearner` package is quite straightforward. At its minimum, a user need only to specify a single dependent variable, a set of predictor variables, and a candidate library of estimators. This minimal option will not provide predictions on an out of sample dataset but will select a discrete super leaner winner, formulate the super learner model (with optimized weights), and perform the super learner cross validation.

Most options to the `superlearner` package are quite customizable, however caution is advised when formulating the candidate estimator library. To demonstrate, say one would like to run `superlearner` on a dependent variable y with predictors a, b, c, d, e. The Stata call could look something like the following:

superlearner *y a b c d e*, **library(***regress glm mixed***)**

This will call the `regress`, `glm`, and `mixed` Stata functions on the aforementioned variables. However, this seems a bit redundant (though valid) and perhaps a more powerful use of the **library** option would look as follows:

superlearner *y a b c d e*, **library(***regress glm mixed custom_a custom_b custom_c***)**

where now we are still calling the same `regress`, `glm`, and `mixed` functions however we have now introduced the addition of three custom models (`custom_a`, `custom_b`, and `custom_c`). The naming of such models must be identified with the naming convention "custom_" and whichever name the user supplies. These custom models could look like the following:

global custom_a = *"lassoregress y a b c"*
global custom_b = *"mixed y a b c d || e: "*
global custom_c = *"chaidforest y a b c d"*

One can quickly see how the use of custom models would be beneficial to the `superlearner` program. Unfortunately, the available machine learning algorithms one can use is quite limited (see previous sections), however as this library of functions grows so will the benefit of the `superlearner` package. Examples of the `superlearner` algorithm are presented in the next chapter using both simulated and actual data.

Results

Depending on what options the user specifies, the `superlearner` package will return, save, and modify a few objects. If an individual runs the basic `superlearner` algorithm, the program will return a discrete super learner winner, make (and store) predictions on the current data set, display the cross validated super learner results, and store the estimated super learner model. If a user chooses to specify an alternative data set to make predictions on (say for instance a test set), then the aforementioned items are returned to the user with the addition of predictions on the out of sample dataset.

Cross Validation Implementation

It should be noted that the framework for cross validation used in the `superlearner` function was heavily modified (with permission) from Ben Daniels' package, `crossFold` [7]. Unfortunately the original `crossFold` package was not customizable enough for this project, however the package did provide a strong foundation for the `superlearner` cross validation methods.

Chapter 5

Examples

Perhaps the best way to see the predictive power of `superlearner` is by observing the algorithm on both a simulated and actual dataset. The benefit of using a simulated dataset allows for methodology validation as the true data distribution is known [18]. One will see through a contrived example how unlikely it would be for any single algorithm in Stata to properly model such a complex relationship between outcome and predictors. In the second example, a real dataset is used in which one may actually want to use the `superlearner` algorithm to make out of sample prediction on a test dataset.

5.1 Simulated Data

Suppose we have a data generating system which comes from the following function:

$$y = 2X \times \mathbf{I}(X <= -2) + X^3 \times \mathbf{I}(-2 < X <= 0) - 2X\mathbf{I}(X > 2) + U \tag{5.1}$$

where $\mathbf{I}$ represents an indicator function and U is some noise generated from a standard normal distribution (an actual plot of the 400 data points is displayed in figure 5.1). Given the scatter plot, one can understand how difficult it would be to fit a single model to appropriately measure the relationship between Y and X (before looking at the data). However, because we are able to use `superlearner` we know that while we are unsure of a single model's ability to correctly depict the relationship between outcome and covariate, we can rest easy knowing that if we were to specify a collection of possible algorithms, `superlearner` will select the optimal combination of such algorithms for the best possible fit.

When using the `superlearner` package, it is absolutely crucial to specify the candidate library of estimators before plotting any of the data. In this example, given the lack of machine learning algorithms available in Stata and the number of predictors (one), the candidate library of estimators chosen was quite scarce. The library involved four linear regression algorithms of which three involved the use of splines. Of course, this simulation is quite contrived and unless the user had a valid and justified reason for using such spline

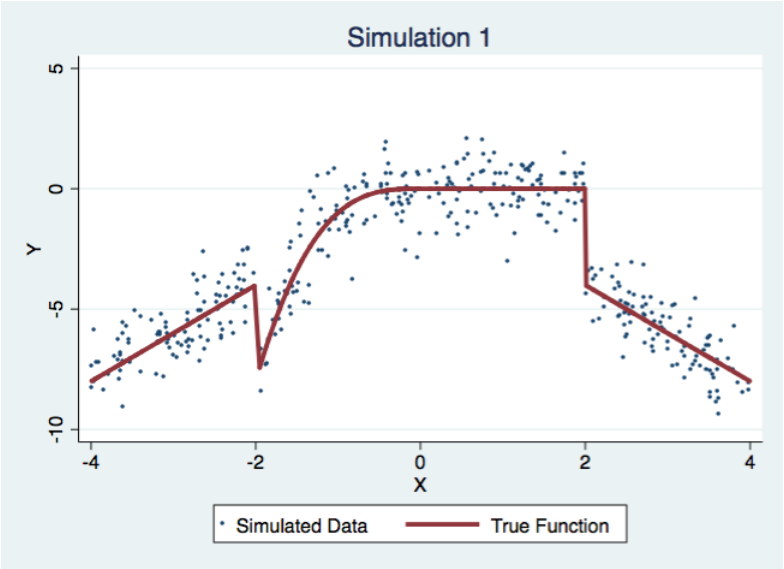

Figure 5.1: Scatterplot for simulation 1. The solid line represents the true relationship between Y and X where the data points represent the simulated data.

construction, this sort of library would not generally exist. Table 5.1 lists the library of estimators and their formulation.

Algorithm Name	Knot Locations (x=)
custom_a	-4, 0 , 4
custom_b	-1, 2
custom_c	1, 2
regress	(not applicable)

Table 5.1: Candidate algorithms used in simulation 1 for modeling the relationship between Y and X.

The first set of results may be seen in Table 5.2 for a summary of the cross validated error and discrete super learner winner (in bold). These results are not entirely surprising; as expected, the standard regression function (regressing y on a single predictor, x) performs the worst amongst all candidate algorithms given the way the data was partitioned into each fold. It is observed that all three custom regression functions perform almost as well as one another, however regression function custom_c has the lowest average cross validated risk across all ten folds. It is important to note that this does not necessarily mean that function custom_c will always be the best candidate algorithm, this simply means given the way the data was split with cross validation, regression model custom_c happened to perform the

best. It is quite possible if we ran this simulation again another candidate algorithm would be selected as the discrete super learner winner.

Algorithm Name	Average MSE
custom_a	2.19
custom_b	2.14
custom_c	**1.74**
regress	9.00

Table 5.2: Average 10-fold cross validation results and discrete super learner winner (custom_c).

Next, the algorithm begins to find the optimal combination of all four algorithms for the best possible super learner fit (Table 5.3). Once the optimal weights have been determined for each of the algorithms, the `superlearner` can then make a prediction on the current data set and cross validate itself. The graphical results of the `superlearner` with fit can be seen in Figure 5.2 where as the cross-validated super learner results (to assess the fit of the algorithm) can be seen in Table 5.4.

Algorithm Name	Optimal Weight
custom_a	.193
custom_b	.139
custom_c	.668
regress	0.00

Table 5.3: Optimal weights as reported by the `superlearner` function. The super learner algorithm is comprised of 19.3% model custom_a, 13.9% model custom_b, and 66.8% model custom_c.

Algorithm Name	Average MSE
custom_a	2.20
custom_b	2.13
custom_c	1.71
regress	9.024
Super Learner	**1.53**

Table 5.4: Average 10-fold cross validation results for the cross validated super learner. The super learner algorithm with the lowest average MSE (1.53).

As one can see from the cross validated super learner results (Table 5.4), the super learner algorithm itself performs the best with the lowest cross validated risk across all ten folds.

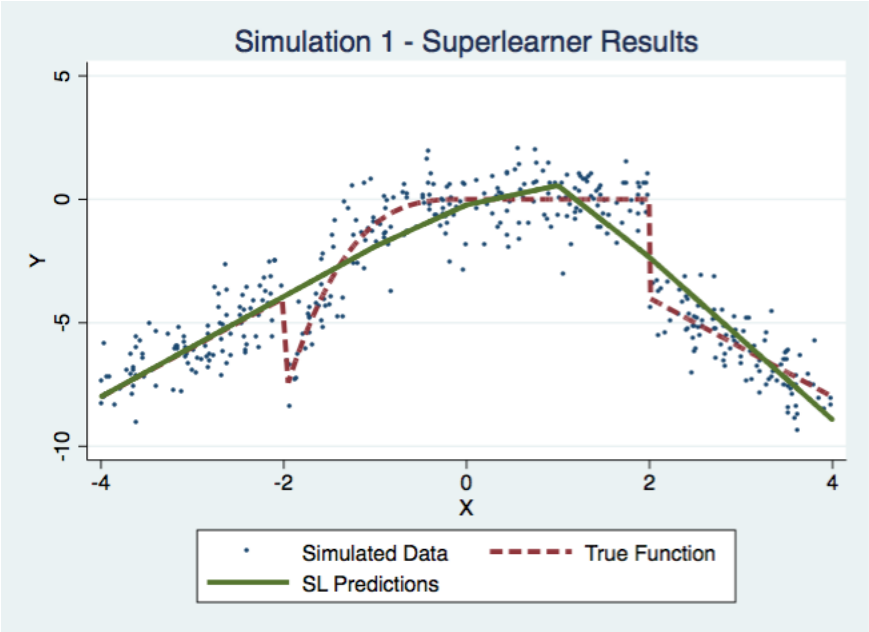

Figure 5.2: Scatterplot of simulated (scattered points), predicted (solid line), and true data (dashed line).

Previously, candidate algorithm custom_c had been selected as our discrete super learner winner however we have just shown that the optimal combination of algorithms custom_a, custom_b, custom_c now performs better than one single algorithm alone.

5.2 Real Data

For this section, a modified version of the available dataset, "House Prices: Advanced Regression Techniques" was used from the Kaggle website [14]. In essence, the House Prices dataset consists of information for 1,460 home sales in the city of Aimes Iowa between the years 2006 and 2009. Along with the sale price of the home, there are 78 additional covariates ranging from the number of square feet to the number of fireplaces within the home contained within the dataset. This example demonstrates the use of `superlearner` by partitioning the data into training and test set for out of sample predictions and comparison across models.

Data Preprocessing

For the sake of this study, only 14 of the 79 covariates were retained from the original dataset for the super learner analysis. The list of covariates ranges from details about the home itself (number of bathrooms, bedrooms, kitchens, fireplaces, square footage, condition of home,

age of home, years since last remodel, garage size, if central air conditioning was installed) to information about the home sale (year house sold and whether the house was sold to a family member or not). The number of home sales in the original data set was 1,460, thus 730 homes were retained for a training sample and 730 homes were set aside as a test set.

Library Selection

The library of estimators chosen for the super learner was restricted to only those models available in Stata. As such, the following models were selected:

- **custom_a:** linear regression, all main terms, and interactions
- **custom_b:** regression with single predictor
- **regress:** linear regression with all main terms
- **lassoregress:** LASSO regression with all covariates
- **ridgeregress:** Ridge Regression with all covariates

It should be noted the previously mentioned available machine learning techniques in Stata proved to be difficult. The package `chaidforest` [20] is only available for classification (not regression) and `svmachines` [11] was unstable (repeatedly shut down the Stata program). While slim, the current library will be sufficient in demonstrating the `superlearner` package.

Results

The initial results of the super learner algorithm can be seen in Table 5.5, which displays the cross validated root mean square error (RMSE) for each of the candidate libraries.

Algorithm Name	Average RMSE
custom_a	246.00
custom_b	236.40
regress	175.07
lassoregress	**174.86**
ridgeregress	236.33

Table 5.5: Average 10-fold cross validation results and discrete super learner winner (lassoregress).

Interestingly, given the way the data was partitioned with $K = 10$ fold cross-validation, lasso regression had the lowest RMSE. It should be noted the particularly large RMSE across all variables is explained by the dependent variable (*salePrice* which ranges from home sale

prices of $34,900 to $755,000. Next, the algorithm began calculating the optimal weights (Table 5.6) used for the final super learner. One may expect the lasso regression model to have the largest weight (considering it had the lowest cross-validated risk) however this is not always the case. In fact, as one can see from Table 5.6, model custom_a had the largest cross validated risk and also the largest weight in our final super learner model.

Algorithm Name	Optimal Weight
custom_a	0.764
custom_b	0
regress	0.236
lassoregress	0
ridgeregress	0

Table 5.6: Optimal weights as reported by the `superlearner` function. Despite the largest cross validated risk estimate, custom_a is displayed with the largest weight in the super learner (.764) followed by the standard regress model (.236).

Next, we could then cross-validated the super learner model itself; the results of the $K = 10$ cross fold validation may be seen in Table 5.7. As one can see, the super learner algorithm itself had the lowest average RMSE of all the models in the candidate library.

Algorithm Name	Average RMSE
custom_a	220.54
custom_b	236.50
regress	171.57
lassoregress	171.34
ridgeregress	236.22
Super Learner	**168.55**

Table 5.7: Average 10-fold cross validation results for the cross validated super learner. It is shown the super learner algorithm has the lowest average RMSE (168.55) and model custom_b with the highest average RMSE (236.50).

Prediction

For the final portion of the analysis, the RMSE for predicted home sale prices versus the actual sale prices were calculated on the held aside test set (Table 5.8). It is noted that the super learner algorithm performed better than any of the other algorithms. Of course, the reported error of the super learner will only decrease as the candidate library evolves, thus these results are promising in showing the predictive power of the super learner algorithm.

Algorithm Name	RMSE
custom_a	273.10
custom_b	861.83
lassoregress	242.62
ridgeregress	867.49
regress	632.42
Super Learner	**207.09**

Table 5.8: RMSE prediction error comparing model results to those in the test set. The super learner algorithm predicts *salePrice* homes in the test set with the lowest RMSE (207.09) and again, custom_b with the highest RMSE of 861.83.

Chapter 6

Discussion

6.1 Difficulties, Limitations, and Current Work in Stata

The presented thesis calls to attention the need for machine learning algorithms in the statistical software, Stata and upon completion, will provide an additional function (`superlearner`) to the Stata community. The main goal of this thesis is three fold: first, to provide a wanted piece of software for the community, secondly, to call to attention the need for modern statistical methods in the Stata programming language, and finally to introduce others who may be unfamiliar or unaware of more modern statistical methodology.

Through a series of examples, this thesis has shown both the customization and functionality of the `superlearner` package in Stata. By using a simulated data set, one is able to see the predictive power of the `superlearner` package in situations where the data distribution is known. In a more practical sense, this thesis also showcases `superlearner`'s ability to make out of sample predictions with a relative high degree of accuracy. The remaining portion of this section will speak to the difficulties, limitations, and future plans for `superlearner`.

6.2 Difficulties/Limitations

After the completion of this project, it is understandable (in part) why perhaps there are a shortage of machine learning algorithms in Stata: the Stata programming language is highly opinionated [6]. In essence, because Stata is a proprietary language there are a consistent set of rules for how functions operate, what they output, and how they are used by the user. This, in and of itself, is both a blessing and a curse; users can easily approach Stata, read the documentation, and understand exactly how to call a specific function. However for the Stata programmer, these constraints can be quite difficult to overcome and can oftentimes limit the functionality of the final software package.

For example, when defining a Stata function (such as `superlearner`), one is required to define the program as one of the following: nclass, rclass, eclass, or sclass depending on what specific type of results the function returns. `superlearner` could very easily fall into an rclass (providing generalized results such as discrete super learner) or eclass (used for generating estimations such as out of sample predictions). Unfortunately one cannot specify multiple classes per Stata function which may cause some confusion to new users.

One of the other difficulties of working with Stata is its ability to only preserve a single data set in memory. Unlike other statistical programming languages (such as SAS and R), Stata is only able to handle and process a single dataset at a time. Normally this is not an issue as one usually performs an analysis on a single dataset, however there are instances in the `superlearner` algorithm where the creation of an additional dataset is necessary (for example, formulation of z-matrix to calculate the optimized weights). Of course, one could simply create an additional dataset however one is not able to reference variables in another dataset unless it is the current open dataset.

This ability to only manipulate a single dataset becomes further complicated by Stata's general practice for chaining function calls. For example, say one would like to make out-of-sample predictions using a linear model. First the user runs the model, next the user simply runs a `predict` command and the predictions are generated. Generally this is not an issue when working with a single model, however once one begins running a series of models, chaining commands back to back becomes incredibly more complicated and difficult to manage. Given the super learner algorithm, it can become quite an arduous task to run a candidate library of fifteen algorithms through a series of ten fold cross validation when there is a constant and consistent chaining of function calls.

However, this is not to say that one should give up on Stata and its ability to implement machine learning packages. Some programmers and bloggers have agreed that at least computationally, Stata is quite proficient compared to other programming languages [12]. When dealing with computationally intense algorithms, Stata could very well be the preferred platform for future machine learning algorithms. Furthermore, while creating programs can be quite difficult for the programmer, the trade off is an improved experience for the people who genuinely matter the most: those who are going to use the software.

6.3 Future Work

The `superlearner` package in Stata (while functional) is far from complete. As of March 2018, the original superlearner package (written for the program R), has six contributors and just under seven years worth of code in the database [22]. Future implementations of the current (Stata) program will hopefully mimic at least most of the functionality of the R package by allowing for: different methods of weight optimization, screening algorithms, graphical outputs, standard errors, cluster identification and perhaps the most important, an improved and expanded set of potential candidate algorithms. Outside of the package

itself, future work will be geared towards a more extensive help guide and the creation of an accompanying GUI.

While the current thesis only makes a small scratch on the surface of the super learner algorithm, it does provide a starting point for others to collaborate. In fact, the current version of `superlearner` can be contributed to via the platform github [5]. Users are encouraged to add features they see necessary or fix existing code in the user base. Hopefully, with enough traction, this package will spark interest in the community and provide (if not document) a valid and practical need for more modern biostatistical methods in Stata sofware.

Bibliography

[1] Leo Breiman. "Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author)". In: *Statistical Science* 16.3 (Aug. 2001), pp. 195–215.

[2] Kenneth P. Burnham and David R. Anderson. *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach*. Springer, 2003. ISBN: 0387953647. URL: https://www.amazon.com/Model-Selection-Multimodel-Inference-Information-Theoretic/dp/0387953647?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0387953647.

[3] Isabel Canette and StataCorp. *Fitting a linear regression with interval (inequality) constraints using nl*. URL: https://www.stata.com/support/faqs/statistics/linear-regression-with-interval-constraints/.

[4] Isabel Canette and StataCorp. *Fitting a regression with interval constraints*. URL: https://www.stata.com/support/faqs/statistics/regression-with-interval-constraints/.

[5] Thomas Carpenito. *A program implementing the super learner algorithm in stata*. 2018. URL: https://github.com/Tommyixi/SuperLearner-STATA.

[6] Nicholas J. Cox. "Suggestions on Stata programming style". In: *The Stata Journal* 5.4 (2005), pp. 560–566.

[7] Benjamin Daniels. *CROSSFOLD: Stata module to perform k-fold cross-validation*. Statistical Software Components S457426. https://ideas.repec.org/c/boc/bocode/s457426.html. Boston College Department of Economics, 2012.

[8] Rahul C. Deo. "Machine Learning in Medicine". In: *Circulation* 132.20 (2015), pp. 1920–1930. ISSN: 0009-7322. DOI: 10.1161/CIRCULATIONAHA.115.001593. eprint: http://circ.ahajournals.org/content/132/20/1920.full.pdf. URL: http://circ.ahajournals.org/content/132/20/1920.

[9] Python Software Foundation. *Index of Packages Matching 'machine learning'*. 1990-2018. URL: https://pypi.python.org/pypi?%3Aaction=search%5C&term=machine+learning%5C&submit=search.

[10] James Gareth et al. *An Introduction to Statistical Learning: with Applications in R (Springer Texts in Statistics)*. Springer, 2017. ISBN: 1461471370.

[11] Nick Guenther and Matthias Schonlau. *Support vector machines*. Statistical Software Components. `https://ideas.repec.org/a/tsj/stataj/v16y2016i4p917-937.html`. StataCorp, 2016.

[12] Murtaza Haider. *Speeding tickets for R and Stata*. 2011. URL: `https://ekonometrics.blogspot.com/2011/04/speeding-tickets-for-r-and-stata.html`.

[13] Kenneth R. Hess. "Graphical methods for assessing violations of the proportional hazards assumption in cox regression". In: *Statistics in Medicine* 14.15 (Aug. 1995).

[14] Kaggle Inc. *House Prices: Advanced Regression Techniques*. URL: `https://www.kaggle.com/c/house-prices-advanced-regression-techniques`.

[15] Mark J. van der Laan, Sandrine Dudoit, and Sunduz Keles. "Asymptotic Optimality of Likelihood Based Cross-Validation". In: *Statistical Applications in Genetics and Molecular Biology* 3.1 (2004).

[16] Mark J. van der Laan, Erin Ledell, and Maya Peterson. "AUC-Maximizing Ensembles through Metalearning". In: *The International Journal of Biostatistics* 12.1 (May 2016), pp. 203–218.

[17] Mark J. van der Laan, Eric C. Polley, and Alan E. Hubbard. "Super Learner". In: *Statistical Applications in Genetics and Molecular Biology* 6.1 (2007).

[18] Mark J. van der Laan and Sherri Rose. *Targeted Learning*. Springer New York, 2011.

[19] StataCorp LLC. *Stata Data Analysis and Statistical Software*. URL: `https://www.stata.com/` (visited on 03/03/2018).

[20] Nicholas Joseph Luchman. *CHAIDFOREST: Stata module to conduct random forest ensemble classification based on chi-square automated interaction detection (CHAID) as base learner*. Statistical Software Components S457932. `https://ideas.repec.org/c/boc/bocode/s457932.html`. Boston College Department of Economics, 2015.

[21] Metacran. *Search results: Machine Learning*. 2018. URL: `https://www.r-pkg.org/search.html?q=Machine+Learning`.

[22] Eric C. Polley. *Current version of the SuperLearner R package*. 2018. URL: `https://github.com/ecpolley/SuperLearner`.

[23] Eric C. Polley et al. *SuperLearner: Super Learner Prediction*. R package version 2.0-22. 2017. URL: `https://CRAN.R-project.org/package=SuperLearner`.

[24] Amandeep Singh, R.K Bansal, and Neetu Jha. "Open Source Software vs Proprietary Software". In: *International Journal of Computer Applications* 14.18 (Mar. 2015).

[25] StataCorp. *Nonlinear combinations of estimators*. Statistical Software Components. `https://www.stata.com/manuals13/rnlcom.pdf`. StataCorp, 2018.

[26] Wilbur Townsend. *ELASTICREGRESS: Stata module to perform elastic net regression, lasso regression, ridge regression*. Statistical Software Components S458397. `https://ideas.repec.org/c/boc/bocode/s458397.html`. StataCorp, 2016.