UC Berkeley
UC Berkeley Previously Published Works

Title
AUC-Maximizing Ensembles through Metalearning.

Permalink
https://escholarship.org/uc/item/3pn6s6zg

Journal
The international journal of biostatistics, 12(1)

ISSN
2194-573X

Authors
LeDell, Erin
van der Laan, Mark J
Petersen, Maya

Publication Date
2016-05-01

DOI
10.1515/ijb-2015-0035

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

HHS Public Access
Author manuscript
Int J Biostat. Author manuscript; available in PMC 2016 August 01.

Published in final edited form as:
Int J Biostat. 2016 May 1; 12(1): 203–218. doi:10.1515/ijb-2015-0035.

AUC-Maximizing Ensembles through Metalearning

Erin LeDell[*],
University of California Berkeley, Division of Biostatistics, Berkeley, CA 94720, USA

Mark J. van der Laan, and
University of California Berkeley, Division of Biostatistics, Berkeley, CA 94720, USA

Maya Peterson
University of California Berkeley, Division of Biostatistics, Berkeley, CA 94720, USA

Abstract

Area Under the ROC Curve (AUC) is often used to measure the performance of an estimator in binary classification problems. An AUC-maximizing classifier can have significant advantages in cases where ranking correctness is valued or if the outcome is rare. In a Super Learner ensemble, maximization of the AUC can be achieved by the use of an AUC-maximining metalearning algorithm. We discuss an implementation of an AUC-maximization technique that is formulated as a nonlinear optimization problem. We also evaluate the effectiveness of a large number of different nonlinear optimization algorithms to maximize the cross-validated AUC of the ensemble fit. The results provide evidence that AUC-maximizing metalearners can, and often do, out-perform non-AUC-maximizing metalearning methods, with respect to ensemble AUC. The results also demonstrate that as the level of imbalance in the training data increases, the Super Learner ensemble outperforms the top base algorithm by a larger degree.

Keywords

AUC; binary classifiation; class imbalance; ensemble learning; machine learning

1 Introduction

In the field of biostatistics, binary classification problems arise in many applications, for example, in diagnostic testing. There are also many problems for which the outcome is rare, or imbalanced, meaning the number of positive cases far outweighs the number of negative cases (or vice versa). In this type of prediction problem, the Area Under the ROC Curve (AUC) is frequently used to measure the performance of an estimator (i. e., model fit) due to its robustness against prior class probabilities. When AUC maximization is the goal, a classifier that aims to specifically maximize AUC can have significant advantages in these types of problems.

[*]**Corresponding author: Erin LeDell,** University of California Berkeley, Division of Biostatistics, Berkeley, CA 94720, USA, ledell@berkeley.edu.

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

However, most commonly used classification algorithms work by optimizing an objective function that is unrelated to AUC – for example, accuracy (or error rate). If the training dataset has an imbalanced outcome, this can lead to classifiers where the majority class has close to 100 % accuracy, while the minority class has an accuracy of closer to 0–10 % [1]. In practice, the accuracy of the minority class is often more important than the accuracy of the majority class. Therefore, unless some type of intervention (e. g., under-sampling, over-sampling) is used to help alleviate this issue, or AUC maximization is inherent to the algorithm, class imbalance may negatively impact the performance of a binary classification algorithm.

In this paper, we introduce an ensemble approach to AUC maximization for binary classification problems. Ensemble methods are algorithms that combine the output from a group of base learning algorithms, with the goal of creating an estimator that has predictive performance over the individual algorithms that make up the ensemble. The Super Learner algorithm [2] is an ensemble algorithm which generalizes stacking [3–5], by allowing for more general loss functions and hence a broader range of estimator combinations. The Super Learner is built on the theory of cross-validation and has been proven to represent an asymptotically optimal system for learning [2].

Super Learner, described in further detail in Section 2, estimates the optimal combination of the base learning algorithms in the ensemble, with respect to a user-defined loss function. The “metalearning step” in the Super Learner algorithm is the process of data-adaptively determining the optimal combination a specific group of base learner fits via a second-level metalearning algorithm. With respect to model performance, this leads to estimators that have superior (or at worst, equal) performance to the top base algorithm in the ensemble. Even if none of the base learners specifically maximize AUC, it is possible to inject AUC-maximization directly into imbalanced data problems via the metalearning step of the Super Learner algorithm.

Any type of parametric or nonparametric algorithm (which is associated with a bounded loss function) can be used in the metalearning step, although in practice, it is common to estimate the optimal linear combination of the base learners. Since the Super Learner framework allows for any loss function (and corresponding risk function), to be used in the metalearning step, it is possible to create ensemble learners that specifically aim minimize a user-defined loss function of interest.

The loss function associated with AUC, also called “rank loss,” measures the bipartite ranking error, or disagreement between pairs of examples. The associated risk is calculated as 1.0 – AUC. In the Super Learner algorithm, minimization of the rank loss or, equivalently, maximization of the AUC, can be approached directly by using an AUC-maximining metalearning algorithm. In Section 3, we discuss how AUC maximization can be formulated as a nonlinear optimization problem. We have implemented the AUC-maximizing metalearning algorithm as an update to the **SuperLearner** R package and demonstrate its usage with a code example.

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

In Section 4, we evaluate the effectiveness of a large number of nonlinear optimization algorithms to maximize the cross-validated (CV) AUC of a Super Learner fit. We compare the cross-validated AUC of the AUC-optimized ensemble fits to the cross-validated AUC of the ensembles that do not attempt to optimize AUC. Super Learner fits using various metalearning algorithms are benchmarked using training sets with varying levels of class imbalance. The results provide evidence that "AUC-maximizing" metalearners can, and often do, out-perform non-"AUC-maximizing" metalearning methods (e. g. non-negative least squares), with respect to cross-validated AUC of the ensemble. The results also demonstrate that, on certain datasets, as the level of class imbalance increases in the response variable of the training set, the Super Learner ensemble out-performs the top base algorithm by a larger degree. This suggests that datasets with greater class imbalance in the outcome variable might benefit from Super Learner ensembles with direct AUC-maximization in the metalearning step.

2 Ensemble metalearning

The Super Learner prediction is the optimal combination of the predicted values from the base learners, which is the motivation behind the name, "Super Learner." The optimal way of combining the base learning algorithms is precisely what is estimated in the metalearning step of the Super Learner algorithm. The output from the base learners, also called "level-one" data in the stacking literature [5], serves as input to the metalearner algorithm. Super Learner theory requires cross-validation to generate the level-one dataset, and in practice, k-fold cross-validation is often used.

The following describes how to construct the level-one dataset. Assume that the training set is comprised of n independent and identically distributed observations, $\{O_1,..., O_n\}$, where $O_i = (X_i, Y_i)$ and $X_i \in \mathbb{R}^p$ is a vector of covariate or feature values and $Y_i \in$ R is the outcome. Consider an ensemble comprised of a set of L base learning algorithms, $\{\psi^1,..., \psi^L\}$, each of which is indexed by an algorithm class, as well as a specific set of model parameters. Then, the process of constructing the level-one dataset will involve generating an $n \times L$ matrix, Z, of k-fold cross-validated predicted values as follows:

1. The original training set, X, is divided randomly into V roughly-equal pieces (validation folds), $X_{(1)},...,X_{(V)}$.
2. For each base learner in the ensemble, ψ^l, k-fold cross-validation is used to generate n cross-validated predicted values associated with the l^{th} learner. These n-dimensional vectors of cross-validated predicted values become the L columns of Z.

The level-one design matrix, Z, along with the original outcome vector, $(Y_1, \ldots, Y_n) \in \mathbb{R}^n$, is then used to train the metalearning algorithm, Φ.

2.1 Base learner library

Super Learner theory does not require any specific level of diversity among the set of base learners, however, a diverse set of base learners (e. g., Linear Model, Support Vector Machine, Random Forest, Neural Net) is encouraged. The more diverse the library is, the

Author Manuscript
Author Manuscript
Author Manuscript
Author Manuscript

greater the ability of the ensemble to approximate the true prediction function. The "base learner library" may also include copies of the same algorithm, indexed by different sets of model parameters. For example, the user can specify multiple Random Forests [6], each with a different splitting criterion, tree depth or "mtry" value.

The base learner prediction functions, $\left\{\hat{\psi}^1, \dots, \hat{\psi}^L\right\}$, are trained by fitting each of the base learning algorithms, $\{\psi^1, \dots, \psi^L\}$, on the training set. The base learners can be any parametric or nonparametric supervised machine learning algorithm. Once the level-one dataset is generated by cross-validating the base learners, the optimal combination of these fits is estimated by applying the metalearning algorithm, Φ, to these data.

2.2 Metalearning algorithms

In the context of Super Learning, the metalearning algorithm is a method that minimizes the cross-validated risk associated with some loss function of interest. For example, if the goal is to minimize mean squared prediction error, the ordinary least squares (OLS) algorithm can be used to solve for $\alpha = (\alpha_1, \dots, \alpha_L)$, the weight vector that minimizes the following:

$$\sum_{i=1}^{n}\left(Y_i - \sum_{i=1}^{L}\alpha_l z_{il}\right)^2$$

In the equation above, z_{il} represents the (i, l) element of the $n \times L$ level-one design matrix, Z. If desired, a non-negativity restriction i. e., $\alpha_l \geq 0$, can be imposed on the weights. There is evidence that this type of regularization increases the predictive accuracy of the ensemble [4]. In this case, the Non-Negative Least Squares (NNLS) algorithm [7] can be used as a metalearner. Both OLS and NNLS are suitable metalearner choices to use when the goal is to minimize squared prediction error. In the **SuperLearner** R package [8], there are five pre-existing metalearning methods available by default, and these are listed in Table 1.

However, in many prediction problems, the goal is to optimize some *objective function* other than the objective function associated with ordinary or nonnegative least squares. For example, in a binary classification problem, if the goal is to maximize the AUC of the model, then an AUC-maximizing algorithm can be used in the metalearning step. Unlike the *accuracy* metric for classification problems, AUC is a performance measure that is unaffected by the prior class distributions [9]. Accuracy-based performance measures implicitly assume that the class distribution of the dataset is approximately balanced and the misclassification costs are equal [10]. However, for many real world problems, this is not the case. Therefore, AUC may be a suitable performance metric to use when the training set has an imbalanced, or rare, binary outcome. Multi-class versions of AUC exist [11, 12], however, we will discuss AUC in the context of binary classification problems.

Although we use AUC-maximization as the primary, motivating example, the technique of targeting a user-defined loss function in the metalearning step can be applied to any bounded loss function, $L(\psi)$. It is worth noting that the loss function, $L(\psi)$, not just risk, $E_0L(\psi)$, must be bounded. The AUC-maximizing metalearning algorithm that we have contributed to the

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

SuperLearner package can be reconfigured so that the Super Learner minimizes any loss function that is possible to implement in code. For binary classification, other performance measures of interest may be *F_1-Score* (or F_β) [13], *Partial AUC* [14, 15], or *H-measure* [10]. A Super Learner ensemble that optimizes any of these metrics can be constructed following the same procedure that we present for AUC-maximization.

3 AUC maximization

Given a set of base learning algorithms, the linear combination of the base learners that maximizes the cross-validated AUC of the Super Learner ensemble can be found using nonlinear optimization.

3.1 Nonlinear optimization

A nonlinear optimization problem is an optimization problem that seeks to minimize (or maximize) some objective function, $f(\alpha)$, where $f:\mathbb{R}^d \to \mathbb{R}, \alpha \in R\mathbb{R}^d$, and the solution space is subject various constraints. To be called "nonlinear," either the function to be maximized or minimized, or at least one of the functions whose value is constrained, must be nonlinear.

Box constraints enforce an upper or lower bound on each of the d optimization parameters. In other words, upper and/or lower bound vectors, $lb = (lb_1,...,lb_d)$ and $ub = (ub_1,...,ub_d)$, can be defined, such that $lb_j \leq a_j \leq ub_j$ for $j = \{1,...,d\}$. In partially constrained, or unconstrained, optimization problems, one or both of these bounds may be $\pm \infty$. There may also exist m separate nonlinear inequality constraints, $f_{cj}(\alpha) \leq 0$ for $j = 1,...,m$, for constraint functions, $f_{cj}(\alpha)$. Lastly, a handful of algorithms are capable of supporting one or more nonlinear equality constraints, which take the form, $h(\alpha) = 0$.

3.1.1 Nonlinear optimization in R—The default optimization methods available in base R (via the optim function), as well as algorithms from the **nloptr** package [16], can be used to approximate the linear combination of the base learners that maximizes the AUC of the ensemble. The default optimization methods in R do not allow for equality constraints such as $\Sigma_j\alpha_j = 1$, therefore normalization of the weights can be performed as an additional step after the optimal weights are determined. Since AUC is a ranking-based measure, normalization of the weights will not change the AUC value. Since normalized weights are more easily interpretable, we normalized the weights as a post-processing step. There are some methods in the **nloptr** package that allow for equality constraints as part of the optimization routine, however, this technique for weight normalization was not used.

The optim function supports general-purpose optimization based on Nelder-Mead [17], quasi-Newton and conjugate-gradient algorithms. There is also a simulated annealing method, i. e., "SANN," [18] however, since this method is quite slow, we did not consider its use as metalearning method to be practical. With the optim function, there is only one method, L-BFGS-B [19, 20], that allows for box-constraints on the weights.

The **nloptr** package is an R interface to the **NLopt** [21] software project from MIT. **NLopt** is an open-source library for nonlinear optimization, which provides a common interface for

Author Manuscript
Author Manuscript
Author Manuscript
Author Manuscript

a number of different optimization routines. We evaluated 16 different global and local optimization algorithms from **NLopt** for the purpose of metalearning. The complete list of **NLopt**-based algorithms is documented in Table 2. A lower bound of 0 is imposed for the weights using the **NLopt** methods to avoid learning negative weights for the base learner contribution. We also evaluated the effect of adding an upper bound of 1 in comparison to leaving the upper bound undefined, i. e., ∞.

3.2 AUC-maximizing ensembles

The **SuperLearner** R package was used to evaluate various metalearning methods. The five pre-existing methods, described in Table 1, were compared against a new set of AUC-maximizing metalearning methods. As part of this exercise, we implemented a new metalearning function, method.AUC, for the **SuperLearner** package. In the method.AUC function, the weights are initialized as $\alpha_{init} = (\frac{1}{L}, \ldots, \frac{1}{L})$, where L is the number of base learners. The function will execute an optimization method that maximizes AUC, and will return the optimal weights found by the algorithm.

The method.AUC function currently supports four optim-based and sixteen **nloptr**-based optimization algorithms by default. Many of these algorithms support box-constraints, so additional configurations of the existing methods can also be specified using our software. An example of how to specify a custom metalearning function by wrapping the method.AUC function is shown in Figure 1. In the example, we specify the metalearner to use the unbounded, optim-based, BFGS method [22–25], with post-optimization normalization of the weights.

A complete list of the default **SuperLearner** metalearning methods plus the new optim and **nloptr**-based AUC-maximizing metalearning methods are listed in Table 2. A total of 37 new AUC-maximizing metalearning functions were evaluated. The AUC_nlopt.X (where X is an integer between 1 and 20) functions implement global optimization routines and the remainder (21–32) are local, derivative-free, methods. The AUC_optim.X (where X is an integer between 1 and 5) functions implement the variations of the optim-based methods.

To reduce the computational burden of evaluating a large number of metalearners, we implemented two functions for the **SuperLearner** package which simplify the processes of re-combining the base learner fits using a new metalearning algorithm. These functions, recombineSL and recombineCVSL, take as input an existing “SuperLearner” or “CV.SuperLearner” fit, and a new metalearning function. They re-use the existing level-one dataset and base learner fits stored in the model object, re-run the metalearning step and return the updated fit. A simple example of how to use the recombineSL function is shown in Figure 3.

4 Benchmark results

This section contains the results from the benchmarks of various metalearning methods in a binary classification problem. The purpose of these benchmarks is twofold. We evaluate whether AUC-maximizing metalearners, compared to other methods such as NNLS, actually maximize the cross-validated AUC of the Super Learner ensemble. We also investigate the

Author Manuscript

Author Manuscript

Author Manuscript

Author Manuscript

training set characteristics that lead to the greatest performance gain of the ensemble (over the best individual model), as measured by cross-validated AUC. In particular, we measure the effect that class imbalance, as well as training size, has on the performance of Super Learner, under various metalearning methods.

After computing the performance of Super Learner using the 42 metalearners under consideration on a dataset, we identify which metalearning algorithm yields the best cross-validated ensemble AUC. The remaining metalearners are compared by calculating the offset, in terms of cross-validated AUC, between the Super Learner utilizing the top metalearing method and the other methods. For example, if the top method produced a model with a CV AUC of 0.75, the offset between the top method and a model with 0.73 CV AUC would be 0.02. The AUC offsets for each dataset-metalearner combination are displayed in a grid-format using heatmaps.

For each training set, we contrast the performance of the best individual model, as determined by a cross-validated grid-search, with the cross-validated AUC of the Super Learner. In the grid search method, the estimator with the best cross-validated performance (as measured by a given loss criterion), is selected as the winning algorithm, and the other algorithms are discarded. In the Super Learning, or stacking, context, the process of generating the level-one dataset using crossvalidation is equivalent to a cross-validated grid-search of the base learners. Moreover, in the Super Learner literature, the grid-search technique is referred to as the Discrete Super Learner algorithm [26].

It is quite common for machine learning practitioners to use a grid-search to evaluate the performance of a set of unique candidate learning algorithms, or unique sets of model parameters within a single algorithm class, as a means to select the best model from those under consideration. Since the metalearning step requires a only small amount of computation as compared to the computation involved in generating level-one data, executing a grid-search and training the Super Learner algorithm are computationally commensurate tasks. Although grid-search and Super Learning require a similar amount of work, the Super Learner ensemble, by optimally combining the set of candidate estimators, can provide a boost in performance over the top base model. For context, we also provide the cross-validated AUC for each of the base models in the ensemble.

In the benchmarks, we use a single, diverse base learner library which contains the following five algorithms: Lasso Regression, Generalized Additive Models, Random Forest (with 1,000 trees), Polynomial Spline Regression and K-Nearest Neighbor ($k = 10$). The R packages that were used for each these methods are listed in Table 3.

4.1 Training datasets

We evaluated twenty different training sets from two sources of data, one from the physics research domain and the other from the medical domain. The first set of datasets have a mid-range cross-validated ensemble AUC (0.72–0.80) and the second group has a lower range of ensemble AUC, (0.60–0.67).

Author Manuscript

Author Manuscript

Author Manuscript

Author Manuscript

4.1.1 HIGGS dataset—This set of benchmarks use training sets derived from the HIGGS dataset [27], a publicly available training set that has been produced using Monte Carlo simulations. The original dataset contains 11 million training examples, 28 numeric features and a binary outcome variable. The associated binary classification task is to distinguish between a background process ($Y = 0$) and a process where new theoretical Higgs bosons are produced ($Y = 1$).

Although this simulated dataset is meant to represent a data generating distribution with a rare outcome (i. e., evidence of the Higgs particle), the class distribution of the outcome variable in the simulated data is approximately balanced, with $P(Y = 1) \approx 0.53$. Two groups of subsets of the HIGGS dataset where created, one with $n = 10,000$ observations and the other with $n = 100,000$ observations. Within each group, ten datasets of fixed size, n, but varying levels of class imbalance, were constructed from the original HIGGS dataset. The following levels of outcome imbalance, $P(Y = 1)$, were evaluated: 1–5 %, 10 %, 20 %, 30 %, 40 % and 50 %.

The **SuperLearner** [8] and **cvAUC** [28] R packages were used to train and cross-validate the AUC of the Super Learner fits. For the $n = 10,000$ sized training sets, 10-fold cross-validation was used to estimate the cross-validated AUC values. Within the Super Learner algorithm, 10-fold cross-validation was used to generate the level-one data, where the folds were stratified by the outcome variable. For the $n = 100,000$ training sets, 2-fold cross-validation was used to estimate cross-validated AUC, and 2-fold, stratified, cross-validation was also used to generate the level-one data.

4.1.2 Diabetes dataset—The second set of benchmarks use training sets derived from the publicly available Diabetes Hospital Readmission dataset [29], sourced from 130 U.S. hospitals during the years, 1999–2008. The original dataset contains 100,000 training examples, 55 predictor columns (a mixture of numeric and categorical features) and a binary outcome variable with $P(Y = 1) \approx 0.46$. The associated binary classification task is to predict hospital readmission among patients known to have diabetes.

In order to simplify the benchmarking process, we used a random subset of the rows and restricted the predictor columns to the numeric columns only. The subset of 8 predictors that were used in our sample is:

- time_in_hospital
- num_lab_procedures
- num_procedures
- num_medications
- number_outpatient
- number_emergency
- number_inpatient
- number_diagnoses

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

The outcome variable, called "readmitted", originally had three outcomes: "< 30", "> 30" and "NO". We recoded the outcome to represent readmission vs no readmission, assigning both the "< 30" and "> 30" examples as the positive response class and "NO" as the negative response class. As compared to the HIGGS dataset, the Diabetes dataset we used contained a relatively weak signal, with cross-validated ensemble AUC values starting around 0.6.

As in the HIGGS example, we created ten datasets of fixed size, $n = 10{,}000$, but varying levels of class imbalance introduced artificially, by randomly subsampling the original dataset in a stratified fashion. The following levels of outcome imbalance, $P(Y = 1)$, were evaluated: 1–5 %, 10 %, 20 %, 30 %, 40 % and 50 %. A second set of larger datasets were also evaluated, with $n = 50{,}000$. 10-fold cross-validation was used to estimate AUC of all of the ensembles.

4.2 Ensemble performance by metalearner

For the HIGGS datasets, the results shown in Tables 4 and 5 suggest that AUC-maximizing metalearners and the negative log-likelihood method typically perform best among all reviewed metalearning algorithms, as measured by cross-validated AUC. Although the AUC-based metalearners often perform better than the non-AUC methods for the HIGGS datasets, there are some examples where using the loss function associated with the negative log-likelihood (of the binomial distribution) yields the top Super Learner ensemble. Unlike the AUC-based methods, the log-likelihood-based metalearning methods in the **SuperLearner** package, such as method.NNloglik and method.CC_nloglik, use radient information in the optimization process, which can be helpful.

4.2.1 HIGGS results

4.2.2 Diabetes results—The Diabetes 10,000-row dataset had slightly different results than the HIGGS datasets, however the 50,000-row Diabetes datasets had results consistent with the HIGGS datasets.

In the smaller ($n = 10{,}000$) set, four of the ten datasets (with different levels of imbalance) that were evaluated achieved top performance among all metalearners using a least-squares based metalearner. This is an interesting and perhaps unexpected result, considering the assumption that AUC-maximization leads to the best results. As shown in Table 6, the dataset with highest level of class imbalance (1 %), the CC_LS and NNLS metalearners performed the best and the log-likihood based metalearner came in next, still offering benefit to using an ensemble over the top single model. As for the remaining AUC-maxiziming metalearning methods, the ensemble did not perform well on any of these – the ensemble performance was actually slightly lower than the performance of GAM. This ensmeble may not represent a typical scenario in terms of performance of the base learners – the range of CV AUCs in the base learner library was 0.5 (Polynomial Spline Regression) – 0.594 (GAM), which are very low AUC values. The rest of the ensembles in the small Diabetes datasets were found to be maximized by the log-likelihood based metalearner. Overall, the cross-validated AUC values for the base learners and the ensemble are relatively low, the highest among all ten datasets being 0.651. As there is a s very low signal in the data, that may be the cause of the contrary results.

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

In the larger ($n = 50,000$) datasets, the AUC-maximization metalearners dominate among other methods. As shown in Table 7, the Super Learner trained on the dataset with the highest imbalance (1 %) benefitted most from the log-likelihood based metalearning method, however all other datasets benefitted most from AUC-maximizing metalearners.

There is not one single metalearning method that performs best across all datasets, so it's recommended to evaluate as many metalearners as possible. Since the metalearning step is rather fast in comparison to any of the L base learning steps, evaluating a large number of metalearners is a reasonable approach to creating the highest performing Super Learner using the available training dataset. Two utility functions were created to facilitate easy re-training of the metalearning step, without having to re-train all the base learners. The functions, now available in the **SuperLearner** package are called recombineSL and recombineCVSL (Figure 4).

4.3 Effect of class imbalance

By creating a sequence of training sets of increasing class imbalance, we evaluated the ability of the Super Learner to outperform individual models. Specifically, we measured the gain, in terms of cross-validated AUC of the ensemble, that the Super Learner provides over the top base model (i. e., the grid-search winner).

What the HIGGS results demonstrate in Table 4, Table 5 and Figure 6 is that the AUC gain achieved by using a Super Learner (SL) ensemble versus choosing the best base algorithm, as selected by grid-search (GS), is highest in the $P(Y = 1) \leq 10$ % range. In particular, for the $P(Y = 1) = 4$ % dataset in Table 4, there is nearly a 0.04 gain in cross-validated AUC achieved from using an AUC-maximizing Super Learner, as opposed to selecting the top base learning algorithm, which in this case is a Generalized Additive Model (GAM). For completeness, the cross-validated grid search performance of each of the base algorithms are provided in Table 8 and Table 9 in the Appendix.

Figure 6 is a plot of the Super Learner AUC gain versus the value for $P(Y = 1)$ for both HIGGS datasets, along with a Loess fit for each group of points. With both datasets, the general trend is that the Super Learner AUC gain increases as the value of $P(Y = 1)$ decreases. However, with the $n = 10,000$ training set, there is a sharp decrease in AUC gain near $P(Y = 1) = 1$ %. This trend is not present in the $n = 100,000$ dataset, so it may be a an artifact of the $n = 10,000$ dataset, due to the global rarity of minority class examples in the training set (there are only 100 minority class observations in total). Future investigation could include examining the level of imbalance with higher granularity, and in particular, in the $P(Y = 1) \leq 1$ % range, using larger n.

The Diabetes results in Table 6, Table 7 and Figure 7 show a similar, though slightly less pronounced, trend. For simple comparison, Figures 6 and 7 are shown on the same scale.

4.4 Effect of regularization on the base learner weights

The odd-numbered **NLopt**-based AUC-maximizing metalearning methods enforce box constraints – weights are bounded below by 0 and above by 1. As shown in Table 2, the even-numbered **NLopt**-based methods are the partially unconstrained counterparts of the

Author Manuscript

Author Manuscript

Author Manuscript

Author Manuscript

odd methods, where the upper bound is ∞. The top metalearner offset heatmap in Figure 4 shows that all of partially unconstrained AUC-maximizing meta-learning methods do poorly in comparison to their fully constrained counterparts.

5 Concluding remarks

In this article, we describe a novel implementation an AUC-maximizing Super Learner algorithm. The new AUC-maximizing metalearning functionality, as well as utility functions for efficient re-estimation of the metalearning algorithm in an existing Super Learner fit, have been contributed to the **SuperLearner** R package.

We benchmarked the AUC-maximizing metalearning algorithms against various existing metalearners that do not target AUC. The results suggest that the use of an AUC-maximizing metalearner in the Super Learner algorithm can, and often does, lead to higher performance, with respect to cross-validated AUC of the ensemble. The benchmarks indicate that Super Learning (which includes the AUC-maximizing meta-learners) is most effective when the outcome is imbalanced – in particular, in the $P(Y = 1) \leq 15\ \%$ range. The benchmarks also show that NNLS, which has been widely used as a metalearning method in stacking algorithms since it was first suggested by Breiman, often performs worse than AUC-maximizing andlog-likelihood based metalearners.

In the current implementation of method.AUC, the weights are initialized as $\alpha_{init} = (\frac{1}{L}, \ldots, \frac{1}{L})$, where L is the number of base learners. In future work, we may investigate the effect of initializing the weights using existing information about the problem, for example, let α_{init} = (0,...,0,1,0,...,0), where the 1 is located at the index of the top base learner, as determined by cross-validation. Further, it may increase performance to use the global optimum as input for a local optimization algorithm, although this technique for chaining algorithms together was not evaluated in our study.

Appendix

Table 8

CV AUC for HIGGS datasets (n = 10,000; CV = 10 × 10).

No.	$P(Y = 1)$	SL	glmnet	gam	randomForest	polymars	knn
1	0.01	0.721	0.675	0.718	0.600	0.504	0.487
2	0.02	0.739	0.667	0.723	0.671	0.530	0.490
3	0.03	0.764	0.692	0.741	0.717	0.500	0.508
4	0.04	0.770	0.686	0.732	0.731	0.579	0.540
5	0.05	0.759	0.675	0.723	0.727	0.569	0.540
6	0.10	0.768	0.672	0.720	0.744	0.750	0.554
7	0.20	0.778	0.679	0.720	0.764	0.760	0.579
8	0.30	0.783	0.679	0.721	0.775	0.757	0.593
9	0.40	0.788	0.682	0.722	0.783	0.763	0.595
10	0.50	0.787	0.683	0.722	0.783	0.756	0.600

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Table 9

CV AUC for HIGGS datasets (n = 100,000; CV = 2 × 2).

No.	$P(Y = 1)$	SL	glmnet	gam	randomForest	polymars	knn
1	0.01	0.754	0.679	0.727	0.708	0.500	0.530
2	0.02	0.766	0.683	0.724	0.729	0.745	0.537
3	0.03	0.772	0.684	0.728	0.742	0.748	0.544
4	0.04	0.777	0.684	0.724	0.749	0.758	0.554
5	0.05	0.779	0.687	0.729	0.759	0.753	0.563
6	0.10	0.786	0.689	0.728	0.772	0.771	0.588
7	0.20	0.793	0.684	0.725	0.786	0.776	0.607
8	0.30	0.795	0.682	0.722	0.792	0.767	0.617
9	0.40	0.798	0.680	0.720	0.796	0.769	0.627
10	0.50	0.800	0.682	0.721	0.798	0.771	0.624

Table 10

CV AUC for diabetes datasets (n = 10,000; CV = 10 × 10).

No.	$P(Y = 1)$	SL	glmnet	gam	randomForest	polymars	knn
1	0.01	0.597	0.527	0.594	0.517	0.500	0.485
2	0.02	0.608	0.593	0.606	0.562	0.500	0.480
3	0.03	0.615	0.597	0.603	0.580	0.500	0.530
4	0.04	0.615	0.581	0.599	0.588	0.500	0.541
5	0.05	0.620	0.588	0.603	0.596	0.500	0.544
6	0.10	0.630	0.603	0.619	0.600	0.500	0.582
7	0.20	0.632	0.615	0.625	0.601	0.533	0.584
8	0.30	0.629	0.616	0.625	0.600	0.581	0.578
9	0.40	0.637	0.623	0.633	0.621	0.609	0.579
10	0.50	0.651	0.641	0.649	0.639	0.616	0.591

Table 11

CV AUC for diabetes datasets (n = 50,000; CV = 10 × 10).

No.	$P(Y = 1)$	SL	glmnet	gam	randomForest	polymars	knn
1	0.01	0.679	0.664	0.673	0.590	0.500	0.543
2	0.02	0.686	0.669	0.684	0.607	0.500	0.549
3	0.03	0.699	0.681	0.696	0.615	0.500	0.561
4	0.04	0.685	0.666	0.681	0.614	0.500	0.567
5	0.05	0.680	0.657	0.672	0.612	0.500	0.571
6	0.10	0.677	0.642	0.658	0.628	0.500	0.596
7	0.20	0.678	0.639	0.655	0.650	0.629	0.617
8	0.30	0.658	0.639	0.649	0.639	0.623	0.603

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

No.	$P(Y = 1)$	SL	glmnet	gam	randomForest	polymars	knn
9	0.40	0.646	0.638	0.644	0.633	0.632	0.584
10	0.50	0.668	0.656	0.660	0.662	0.644	0.609

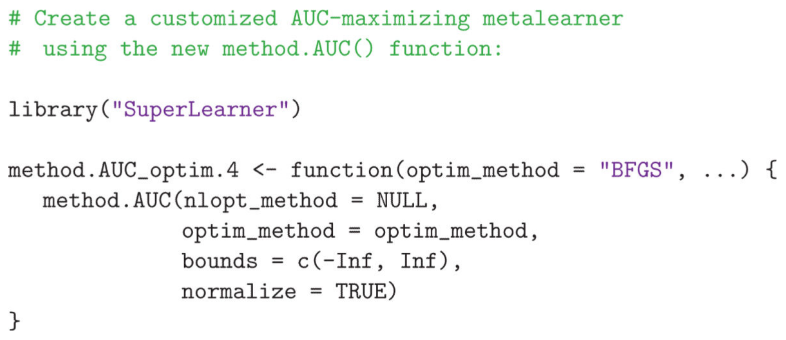
References

1. Woods K, Doss C, Bowyer K, Solka J, Priebe C, Kegelmeyer W. Comparative evaluation of pattern recognition techniques for detection of microcalcifications in mammography. Int J Pattern Recognit Artif Intell. 1993; 7:1417–36.
2. van der Laan M-J, Polley E-C, Hubbard A-E. Super learner. Stat Appl Genet Mol Biol. 2007:6.
3. LeBlanc M, Tibshirani R. Combining estimates in regression and classification, Technical report. J Am Stat Ass. 1993
4. Breiman L. Stacked regressions. Mach Learn. 1996; 24:49–64.
5. Wolpert D-H. Stacked generalization. Neural Networks. 1992; 5:241–59.
6. Breiman L. Random forests. Mach Learn. 2001; 45:5–32.
7. Lawson CL, Hanson RJ. Solving least squares problems. SIAM. 1974
8. Polley, E.; van der Laan, M. SuperLearner: super learner prediction. r package version 2.0–9. 2010. Available at: http://CRAN.R-project.org/package=SuperLearner
9. Bradley AP. The use of the area under the roc curve in the evaluation of machine learning algorithms. Pattern Recognit. 1997; 30:1145–59.
10. Hand D. Measuring classifier performance: a coherent alternative to the area under the roc curve. Mach Learn. 2009; 77:103–23.
11. Hand D, Till R. A simple generalisation of the area under the roc curve for multiple class classification problems. Mach Learn. 2001; 45:171–86.
12. Provost, F.; Domingos, P. Well-trained pets: improving probability estimation trees. Stern School of Business, New York Univ; 2000. CeDER Working Paper: IS-00-04
13. Rijsbergen, CJV. Information Retrieval. 2 ed.. Butterworth; London, UK: 1979.
14. McClish DK. Analyzing a portion of the roc curve. Med Decis Making. 1989; 9:190–5. [PubMed: 2668680]
15. Jiang Y, Metz CE, Nishikawa RM. A receiver operating characteristic partial area index for highly sensitive diagnostic tests. Radiology. 1996; 201:745–50. [PubMed: 8939225]
16. Ypma, J.; Borchers, H-W.; Eddelbuettel, D. R interface to NLopt. version 1.0.4. 2014. Available at: http://cran.r-project.org/web/packages/nloptr/index.html
17. Nelder JA, Mead R. A simplex method for function minimization. The Computer Journal. 1965; 4:308–13.
18. Belsie CJP. Convergence theorems for a class of simulated annealing algorithms on rd. J Appl Probab. 1992; 29:885–92.
19. Byrd RH, Lu P, Nocedal J. A limited-memory algorithm for bound-constrained optimization. 1995
20. Zhu C, Byrd R-H, Lu P, Nocedal J. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. ACM Trans Math Softw. 1997; 23:550–60. Available at: http://doi.acm.org/10.1145/279232.279236.
21. Johnson, SG. The NLopt nonlinear-optimization package. 2014. Available at: http://ab-initio.mit.edu/nlopt
22. Broyden CG. The convergence of a class of double-rank minimization algorithms. J Inst Math Appl. 1970; 6:76–90.
23. Fletcher R. A new approach to variable metric algorithms. Comput J. 1970; 13:317–22.
24. Goldfarb D. A family of variable metric updates derived by variational means. Math Comput. 1970; 24:23–6.
25. Shanno DF. Conditioning of quasi-newton methods for function minimization. Math Comput. 1970; 24:647–56.

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

26. van der Laan, M-J.; Polley, E-C. Super learner in prediction, U.C. Berkeley Division of Biostatistics Working Paper Series. 2010. Available at: http://biostats.bepress.com/ucbbiostat/paper266
27. Baldi, P.; Sadowski, P.; Whiteson, D. Searching for exotic particles in high-energy physics with deep learning.; Nat Commun. 2014. p. 5Available at: http://archive.ics.uci.edu/ml/datasets/HIGGS
28. LeDell E, Petersen M, van der Laan M. cvAUC: Cross-validated area under the ROC curve confidence intervals. r package version 1.1. 0:2015. Available at: http://CRAN.R-project.org/package=cvAUC.
29. Strack, B.; DeShazo, JP.; Gennings, C.; Olmo, JL.; Ventura, S.; Cios, KJ., et al. Impact of hba1c measurement on hospital readmission rates: Analysis of 70,000 clinical database patient records.. BioMed Res Int. 2014. Available at: http://archive.ics.uci.edu/ml/datasets/Diabetes + 130-US + hospitals + for + years + 1999-2008

Author Manuscript Author Manuscript Author Manuscript Author Manuscript



```
# Create a customized AUC-maximizing metalearner
#  using the new method.AUC() function:

library("SuperLearner")

method.AUC_optim.4 <- function(optim_method = "BFGS", ...) {
   method.AUC(nlopt_method = NULL,
              optim_method = optim_method,
              bounds = c(-Inf, Inf),
              normalize = TRUE)
}
```

Figure 1.

Example of how to use the method. AUC function to create customized AUC-maximizing metalearning functions.

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

```
# Example base learner library:
SL.library <- list(c("SL.glmnet"),
                   c("SL.gam"),
                   c("SL.randomForest"),
                   c("SL.polymars"),
                   c("SL.knn"))

# Then the "method.AUC_optim.4" function can be used as follows:
fit <- SuperLearner(Y = Y, X = X, newX = newX,
                    family = binomial(),
                    SL.library = SL.library,
                    method = "method.AUC_optim.4")
```

Figure 2.

Example of how to use the custom method. AUC_optim.4 metalearning function with the SuperLearner function.

```
# Assume that ‘fit’ is an object of class, "SuperLearner"
newfit <- recombineSL(object = fit, Y = Y, method =
    "method.NNloglik")
```

Figure 3.
Example of how to update an existing “SuperLearner” fit by re-training the metalearner with a new method.

Author Manuscript
Author Manuscript
Author Manuscript
Author Manuscript

Author Manuscript

Author Manuscript

Author Manuscript

Author Manuscript

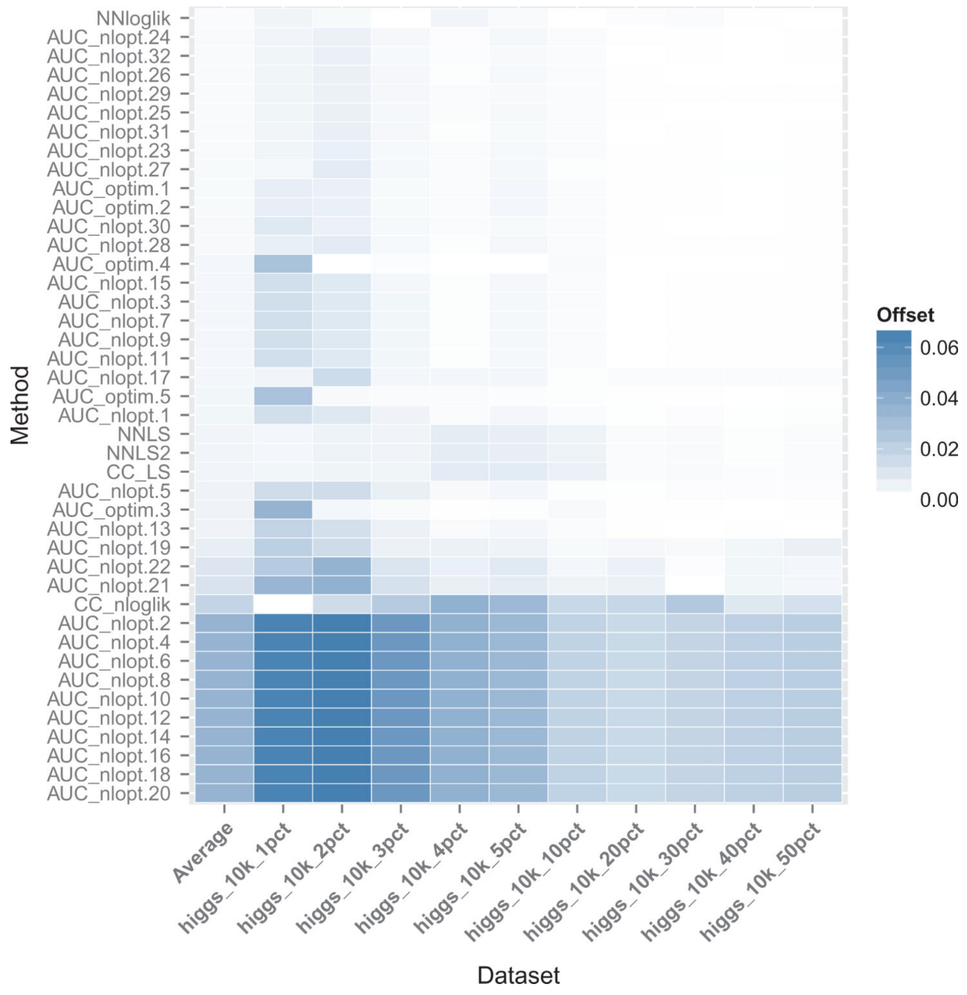


Figure 4.

Model CV AUC offset from the best Super Learner model for different metalearning methods. (No color = = top metalearner).

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

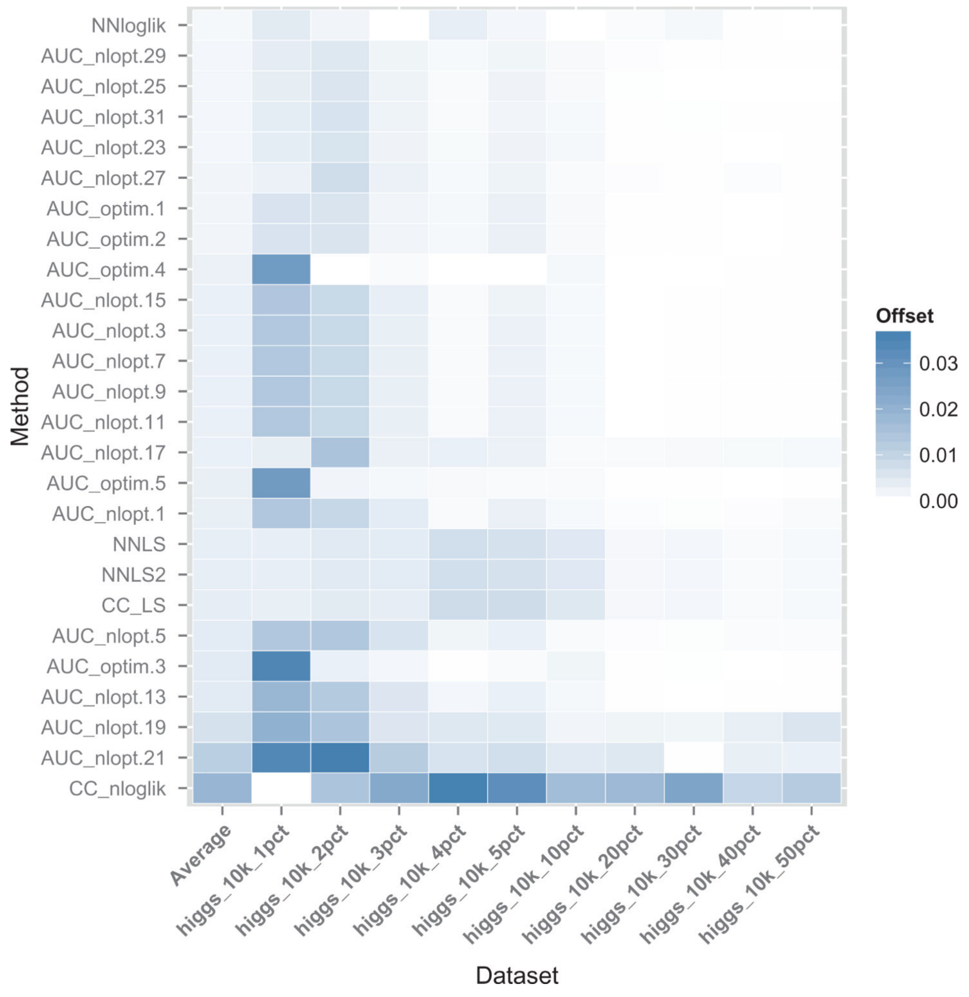


Figure 5.

Model CV AUC offset from the best Super Learner model for a subset of the metalearning methods. The partially unconstrained NLopt methods are removed in order to more clearly demonstrate the differences between the box-constrained methods. (No color == top metalearner).

Author Manuscript

Author Manuscript

Author Manuscript

Author Manuscript

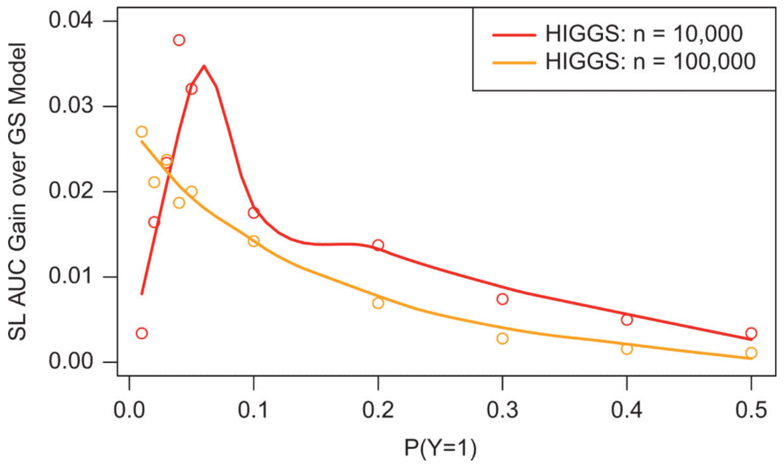


Figure 6.
CV AUC gain by super learner (with top metalearner) over grid-search winning model on HIGGS datasets. Loess fit overlays actual points.

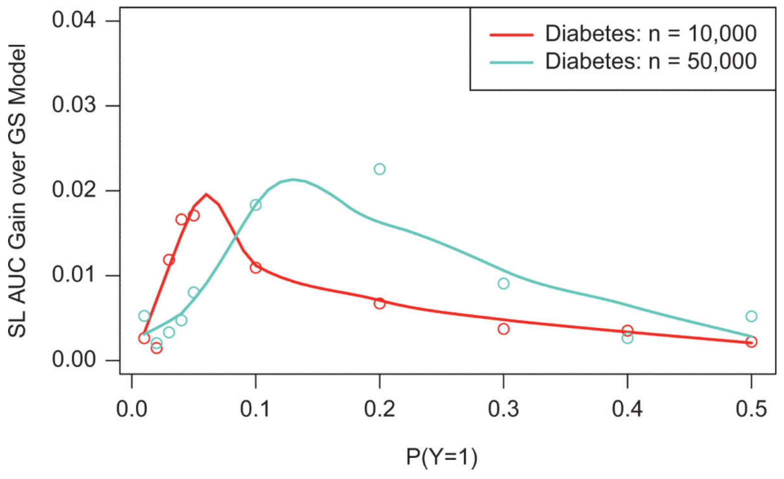


Figure 7.

CV AUC gain by super learner (with top metalearner) over grid-search winning model in diabetes datasets. Loess fit overlays actual points.

Author Manuscript
Author Manuscript
Author Manuscript
Author Manuscript

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Table 1

Default metalearning methods in **SuperLearner** R package.

Method	Description	R Package
NNLS	Non-negative least squares	nnls
NNLS2	Non-negative least squares	quadprog
CC_LS	Non-negative least squares	nloptr
NNloglik	Negative log-likelihood (Binomial)	Base R
CC_nloglik	Negative log-likelihood (Binomial)	nloptr

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Table 2

Metalearning methods evaluated.

Method	Description	min{α_j)	max{α_j}
NNLS	Non-negative least squares (nnls)	0	Inf
NNLS2	Non-negative least squares (quadprog)	0	Inf
CC_LS	Non-negative least squares (nloptr)	0	1
NNloglik	Negative log-likelihood (Binomial) (optim)	0	Inf
CC_nloglik	Negative log-likelihood (Binomial) (nloptr)	0	1
AUC_nlopt.1	DIRECT	0	1
AUC_nlopt.2	DIRECT	0	Inf
AUC_nlopt.3	DIRECT-L	0	1
AUC_nlopt.4	DIRECT-L	0	Inf
AUC_nlopt.5	DIRECT-L RAND	0	1
AUC_nlopt.6	DIRECT-L RAND	0	Inf
AUC_nlopt.7	DIRECT NOSCAL	0	1
AUC_nlopt.8	DIRECT NOSCAL	0	Inf
AUC_nlopt.9	DIRECT-L NOSCAL	0	1
AUC_nlopt.10	DIRECT-L NOSCAL	0	Inf
AUC_nlopt.11	DIRECT-L RAND NOSCAL	0	1
AUC_nlopt.12	DIRECT-L RAND NOSCAL	0	Inf
AUC_nlopt.13	ORIG DIRECT	0	1
AUC_nlopt.14	ORIG DIRECT	0	Inf
AUC_nlopt.15	ORIG DIRECT-L	0	1
AUC_nlopt.16	ORIG DIRECT-L	0	Inf
AUC_nlopt.17	Controlled Random Search with Local Mutation	0	1
AUC_nlopt.18	Controlled Random Search with Local Mutation	0	Inf
AUC_nlopt.19	Improved Stochastic Ranking Evolution Strategy	0	1
AUC_nlopt.20	Improved Stochastic Ranking Evolution Strategy	0	Inf
AUC_nlopt.21	Principal Axis (PRAXIS)	0	1
AUC_nlopt.22	Principal Axis (PRAXIS)	0	Inf
AUC_nlopt.23	Constrained Opt. by Linear Approximations	0	1
AUC_nlopt.24	Constrained Opt. by Linear Approximations	0	Inf
AUC_nlopt.25	Bounded NEWUOA	0	1
AUC_nlopt.26	Bounded NEWUOA	0	Inf
AUC_nlopt.27	Nelder-Mead	0	1
AUC_nlopt.28	Nelder-Mead	0	Inf
AUC_nlopt.29	Sbplex	0	1
AUC_nlopt.30	Sbplex	0	Inf
AUC_nlopt.31	BOBYQA	0	1
AUC_nlopt.32	BOBYQA	0	Inf
AUC_optim.1	L-BFGS-B	0	1
AUC_optim.2	L-BFGS-B	0	Inf

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Method	**Description**	**min{α_j)**	**max{α_j}**
AUC_optim.3	Nelder-Mead	–Inf	Inf
AUC_optim.4	BFGS	–Inf	Inf
AUC_optim.5	Conjugate Gradient (CG)	–Inf	Inf

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Table 3

Example base learner library representing a small, yet diverse, collection of algorithm classes. Default model parameters were used.

	Algorithm	R Package	Function
ψ^1	Lasso regression	glmnet	glmnet
ψ^2	Generalized additive model	gam	gam
ψ^3	Random forest (1,000 trees)	random Forest	randomForest
ψ^4	Polynomial spline regression	polyspline	polymars
ψ^5	K-Nearest neighbor ($k = 10$)	class	knn

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Table 4

Top Metalearner performance for HIGGS Datasets, as measured by cross-validated AUC (n = 10,000; CV = 10 × 10).

No.	$P(Y = 1)$	Metalearner	SL AUC Gain	SL AUC	GS AUC	GS Model
1	0.01	CC_nloglik	0.003	0.721	0.718	gam
2	0.02	AUC_optim.4	0.016	0.739	0.723	gam
3	0.03	NNloglik	0.023	0.764	0.741	gam
4	0.04	AUC_optim.4	0.038	0.770	0.732	gam
5	0.05	AUC_optim.4	0.032	0.759	0.727	randomForest
6	0.10	NNloglik	0.018	0.768	0.750	polymars
7	0.20	AUC_nlopt.11	0.014	0.778	0.764	randomForest
8	0.30	AUC_optim.4	0.007	0.783	0.775	randomForest
9	0.40	AUC_nlopt.23	0.005	0.788	0.783	randomForest
10	0.50	NNloglik	0.003	0.787	0.783	randomForest

SL = “Super Learner” and GS = “Grid Search.”

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Table 5

Top Metalearner performance for HIGGS datasets, as measured by cross-validated AUC (n = 100,000; CV = 2 × 2).

No.	$P(Y = 1)$	Metalearner	SL AUC Gain	SL AUC	GS AUC	GS Model
1	0.01	NNloglik	0.027	0.754	0.727	gam
2	0.02	AUC_nlopt.19	0.021	0.766	0.745	polymars
3	0.03	AUC_optim.5	0.024	0.772	0.748	polymars
4	0.04	AUC_optim.4	0.019	0.777	0.758	polymars
5	0.05	AUC_optim.3	0.020	0.779	0.759	randomForest
6	0.10	AUC_nlopt.13	0.014	0.786	0.772	randomForest
7	0.20	AUC_nlopt.5	0.007	0.793	0.786	randomForest
8	0.30	AUC_nlopt.13	0.003	0.795	0.792	randomForest
9	0.40	AUC_nlopt.21	0.002	0.798	0.796	randomForest
10	0.50	AUC_nlopt.11	0.001	0.800	0.798	randomForest

SL = "Super Learner" and GS = "Grid Search."

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Table 6

Top Metalearner performance for diabetes datasets, as measured by cross-validated AUC (n = 10,000; CV = 10 × 10).

No.	$P(Y = 1)$	Metalearner	SL AUC Gain	SL AUC	GS AUC	GS Model
1	0.01	CC_LS	0.003	0.597	0.594	gam
2	0.02	AUC_optim.3	0.001	0.608	0.606	gam
3	0.03	NNloglik	0.012	0.615	0.603	gam
4	0.04	NNloglik	0.017	0.615	0.599	gam
5	0.05	NNloglik	0.017	0.620	0.603	gam
6	0.10	NNLS	0.011	0.630	0.619	gam
7	0.20	NNLS	0.007	0.632	0.625	gam
8	0.30	NNLS	0.004	0.629	0.625	gam
9	0.40	NNloglik	0.004	0.637	0.633	gam
10	0.50	NNloglik	0.002	0.651	0.649	gam

SL = “Super Learner” and GS = “Grid Search.”

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Table 7

Top Metalearner performance for diabetes datasets, as measured by cross-validated AUC (n = 50,000; CV = 10 × 10).

No.	$P(Y = 1)$	Metalearner	SL AUC Gain	SL AUC	GS AUC	GS Model
1	0.01	NNloglik	0.006	0.679	0.673	gam
2	0.02	AUC_optim.5	0.002	0.686	0.684	gam
3	0.03	AUC_optim.5	0.003	0.699	0.696	gam
4	0.04	AUC_optim.3	0.004	0.685	0.681	gam
5	0.05	AUC_optim.3	0.008	0.680	0.672	gam
6	0.10	AUC_optim.4	0.019	0.677	0.658	gam
7	0.20	AUC_optim.4	0.023	0.678	0.655	gam
8	0.30	AUC_optim.4	0.009	0.658	0.649	gam
9	0.40	AUC_optim.4	0.002	0.646	0.644	gam
10	0.50	AUC_optim.4	0.008	0.668	0.660	gam

SL = "Super Learner" and GS = "Grid Search."