A General Ensemble Learning Framework for Online and Offline Machine Learning

by

Cheng Ju

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 Mark van der Laan, Chair
Professor Maya L. Petersen
Professor Haiyan Huang

Spring 2016

Abstract

A General Ensemble Learning Framework for Online and Offline Machine Learning

by

Cheng Ju

Master of Arts in Biostatistics

University of California, Berkeley

Professor Mark van der Laan, Chair

Ensemble learning methods train several baseline models, and use some techniques to combine them together to make prediction. The ensemble learning methods have gained popularity because their superior prediction performance. The optimal learner for prediction varies greatly depending on the underlying data-generating distribution. To select the best algorithm for a given set of data we must therefore use cross-validation to compare several candidate algorithms. Super learner (SL) is an ensemble learning algorithm that utilizes the strength of individual methods by building, training, and selecting among a "library" of candidate algorithms. It can easily be adapted to different data sets by utilizing a diverse library of algorithms.

We first study the offline Super Learner in the setting of propensity score prediction. Propensity scores are used to reduce the bias of causal effect estimators introduced in observational studies by the nonrandom assignment of treatment. The propensity score is the estimated probability of treatment assignment, conditional on baseline covariates; generating propensity scores can be seen as a classic prediction problem. Parametric models like logistic regression and machine learning algorithms including decision trees, neural networks, and support vector machines can all be used for propensity score prediction. The High-dimensional Propensity Score Adjustment (hdPS) developed by Schneeweiss et al. is one parametric algorithm that employs a nonparametric variable selection method to enhance the prediction precision. We compare and combine Super learner and the High-dimensional Propensity Score Adjustment algorithm to predict propensity scores. Prediction performance is assessed across three datasets using three metrics: likelihood, area under the curve, and time complexity. The results show: 1. Without overfitting, hdPS often outperforms other algorithms that only use baseline variables. 2. The best individual algorithm is highly dependent on the data set. An ensemble data-adaptive method like SL is necessary to reliably generate the best estimation. 3. SL which utilizes the hdPS methodology outperforms all other algorithms considered in this study. 4. Moreover, in our study, the results show the reliability of SL: Even though we used likelihood to train the Super Learners, SL still performs best with respect to area under the curve (AUC) in all three data sets.

Then we study the Super Learner in the online setting. Consider an online classification problem. The incoming data stream could be heterogeneous, and even adversarial, e.g. email spamming. We have m base learner/experts which estimate the probability for the incoming data. We want to find a convex combination of the base learners. We use the logistic loss as the surrogate convex loss. This could be treated as an online convex optimization problem. First we show this could be solved by mirror descent, which could achieve $\mathcal{O}(\sqrt{n})$ regret under a mild assumption. Then we show the loss function is α-log-concave and adapt the online Newton steps by Hazan et al to solve the online Super Learner with logarithm regret. To achieve better practical performance, we also study the deeper non-linear Super Learner, which ensembles the predictors in the logit scale. The performance of the online Super Learner are compared to other ensemble methods (e.g. online bagging) on the simulations and real data sets.

To my family and friends.

Contents

List of Figures

List of Tables

Acknowledgments

The author wishes to acknowledge several individuals whose support made this work possible. Special thanks to committee chair Professor Mark van der Laan for his advice and guidance. Thanks also to committee members Professor Maya Petersen and Professor Haiyan Huang for their assistance and comments. My deepest thanks also goes to Professor Alan Hubbard, Professor Nicholas Jewell, Professor Lexin Li for the great lecture and guidance, which offer me a deeper understanding of statistics. I am also thankful to Samuel Lendle and Mary Combs for the helpful comments and efforts in the experiments and writing. I also owe many thanks to Sharon Norris, who helped me solve a lot of administrative issues.

Finally, sincere thanks to family and friends, especially Hua Cheng and Mengxin Ji, for their invaluable encouragement and support.

Chapter 1

Prediction of propensity score using Super Learner and High-dimensional Propensity Score Adjustment

Introduction

The propensity score is the conditional probability of assignment to a particular treatment given a vector of observed covariates. Both large and small sample size theories show that adjustment for the scalar propensity score is sufficient to remove the bias of average treatment effect due to baseline confounders [30]. Therefore estimating the propensity score is a significant step toward efficiently estimating a causal effect. In this article, we aim to compare several powerful algorithms for the estimation of propensity scores.

We focus on developing a single best model to predict propensity scores for binary treatments. One traditional approach to this estimation problem is to use logistic regression [6]. Many modern statistical and machine learning methods can also be used for this problem including classification tree, boosting, and random forest [16]. To predict a binary outcome given a set of covariates, one has many estimation procedures in their toolbox; however, every model has its own bias-variance trade off and various tuning parameters to consider. Therefore, one method may be better than others within a particular data set, while fail with different data sets. It is challenging to figure out the best single model for this particular problem.

Super Learner is a general loss-based learning method that has been proposed and analyzed theoretically in van der Laan et al [21]. It is an ensemble learning algorithm which creates a weighted combination of many candidate learners to build the optimal estimator. In the present study we used the negative log likelihood loss function to inform the algorithm selection. V-fold cross-validation is used to compare performance of the library of candidate estimators. For each candidate algorithm, SL averages the estimated risks across the validation sets, resulting in the so-called cross-validated risk. Cross-validated risk estimates are used to choose the weighted linear convex combination of the candidate learners with the smallest estimated risk [24]. It has been demonstrated [38, 8, 37] that this convex combination performs asymptotically at least as well as the best choice among the library of candidate algorithms if the library does not contain a correctly specified parametric model; Otherwise, it achieves the same rate of convergence as the correctly specified parametric model.

The High-dimensional propensity score algorithm (hdPS) was proposed to automate confounding adjustment for problems involving large healthcare databases. In this setting expert selection of confounders by hand is not practical because of the high dimensionality and sparsity of these databases [33]. Claims data contain information about patient health status. These sparse covariates are often only significant when combined with other covariates. hdPS is a simple and powerful algorithm to explore the utility of this additional information, and previous articles demonstrate hdPS effectively handles baseline confounding through matching and covariance adjustment [29, 36, 28].

In the present study, we compared several statistical and machine learning classification algorithms, the High-dimensional propensity score algorithm, and Super Learners with different libraries on three datasets. Performances of algorithms are assessed by time complexity, negative log-likelihood and AUC (Area Under Curve). In this study, we focus on the estimation of the propensity score (the probability of exposure), and omit consideration of the outcome variable, except when using hdPS (see details in Methods section).

Data Sources and Study Cohorts

We used three data sets to assess the performance of the models: the Novel Oral Anticoagulant Prescribing (NOAC) data set, the Nonsteroidal anti-inflammatory drugs (NSAID) data set and the Vytorin data set. For each data set, we had two kinds of covariates: baseline covariates and claims code covariates. Baseline covariates include demographic (e.g. age, sex, census region and race) and some predefined covariates that are selected by context knowledge. Claims code covariates were collected from extremely high-dimensional and sparse healthcare databases.

Novel Oral Anticoagulant (NOAC) data set

The NOAC data set was generated to track a cohort of new users of oral anticoagulants for use in a study of the comparative safety and effectiveness of these agents. The data was collected by United Healthcare, recorded between October, 2009 and December, 2012. The dataset includes 18,447 observations, 60 baseline covariates and 23,531 claims code covariates.

Each claims code covariate records the number of times a claims code occurred for each patient. The claims code covariates fall into four categories, or ”dimensions”: inpatient diagnoses, outpatient diagnoses, inpatient procedures and outpatient procedures.

for example, if a patient has a value of 2 for the variable "pxop_V5260", then the patient received the outpatient procedure coded as V5260 twice between October, 2009 and December, 2012.

NSAID data set

The NSAID data set tracks the binary exposure to a selective COX-2 inhibitor or a comparison drug, a nonselective NSAID. The observations are drawn from a population of patients aged 65 years and older enrolled in both Medicare and the Pennsylvania Pharmaceutical Assistance Contract for the Elderly (PACE) programs between 1995 and 2002. There are 49,653 observations, with 22 baseline covariates and 9,470 claims code covariates in this study.

Each claims code covariate records the number of times a claims code occurred for each patient. The claims code covariates fall into eight dimensions: prescription drugs, ambulatory diagnoses, hospital diagnoses, nursing home diagnoses, ambulatory procedures, hospital procedures, doctor diagnoses and doctor procedures.

Vytorin data set

This data set was generated to track a cohort of new users of Vytorin and high-intensity statin therapies. The observation is all United Healthcare patients linked for January 1, 2003 December 31, 2012, with age over 65 on day of entry into cohort.

The binary exposure variable tracks use of Vytorin or high-intensity statins. The dataset includes 148,327 observations, 67 baseline covariates and 15,010 code covariates.

Each claims code covariate records the number of times a claims code occurred for each patient. The claims code covariates fall into five dimensions: ambulatory diagnoses, ambulatory procedures, prescription drugs, hospital diagnoses and hospital procedures.

Methods

In this paper, we used R (version 3.2.2) for the data analysis. For each dataset, we randomly selected 80% of the data as the training set and the rest as the testing set. We centered and scaled each of the covariates. We used cross-validation on only the training set to select tuning parameter values for the relevant algorithms, and assessed the goodness of fit of all the models on only the testing set to ensure objective measures of prediction reliability.

High-dimensional propensity score (hdPS) algorithm

We followed the five steps in the hdPS screening method to generate the hdPS covariates:

1. **Specifying Data Resource** First, we clustered the data by their resources, or dimensions, e.g. diagnoses, procedures, and medications. See dataset specific details above.

2. Identifying Candidate Empirical Covariates Second, we empirically identified candidate covariates in each dimension. Within each data dimension we identified the n most prevalent covariates. Here we defined prevalence as the minimum of Pr and $1 - Pr$, where Pr is the proportion of non-zero values for a given covariate.

3. Assessing Recurrence of Code For each selected covariate, e.g. X, we defined three dummy covariates to indicate if that code appeared at least once, at least more than the median, and at least more than the 75^{th} percentile. Then we kept only the dummy covariates.

4. Prioritizing Covariates For each covariate, we computed the potential amount of confounding the variable could adjust for. In a multiplicative model with binary exposure and outcome, a standard measure of confounding adjustment is $Bias_M$:

$$Bias_M = \frac{P_{C1}(RR_{CD} - 1) + 1}{P_{C0}(RR_{CD} - 1) + 1}, \text{ If } RR_{CD} > 1$$

$$Bias_M = \frac{P_{C1}(\frac{1}{RR_{CD}} - 1) + 1}{P_{C0}(\frac{1}{RR_{CD}} - 1) + 1}, \text{ If } RR_{CD} < 1$$

Details can be found in [33]. Here $P_{C1} = P(C = 1|A = 1)$, $P_{C0} = P(C = 0|A = 1)$, $RR_{CD} = \frac{P(Y=1|C=1)}{P(Y=1|C=0)}$

5. Selecting Covariates for Adjustment We select the top k empirical covariates from step 4.

After this five step data screening process the hdPS algorithm estimates the exposure propensity score using multivariate logistic regression, including the k screened hdPS covariates, d demographic covariates and l predefined covariates that were selected by context knowledge. The tuning parameters of this algorithm are the number of claims codes per dimension defined in step 1, n, and the number of screened hdPS covariates in total, k.

For clarification, we call the whole procedure, including screening and estimation the hdPS algorithm, and the screening steps the hdPS screening method.

Machine Learning Algorithm Library

There are many different machine learning algorithms in R with varying syntax for model training and/or prediction across packages. The `caret` package (version 6.0-37) [20] offers wrapper functions that standardized input and output formatting. In this paper, we made a wrapper function to access the machine learning algorithms in the `caret` package and used the `caret` wrapper functions to build the Super Learner libraries.

For each individual algorithm, we used leave group out (LGO) cross-validation to select the tuning parameters. We randomly selected 90% of the training data for model training and 10% of the training data for model tuning and selection. For clarity, we refer to these subsets of the training data as the LGO training set and the LGO validation set, respectively. After the tuning parameters are selected, we fit the selected models on the whole training set, and assess the models on the testing set.

We use 23 statistical/machine learning algorithms in the `caret` package: Bayesian Generalized Linear Model (”bayesglm”), C5.0 (”C5.0”), Single C5.0 Ruleset (”C5.0Rules”), Single C5.0 Tree (”C5.0Tree”), Conditional Inference Tree (”ctree2”), Multivariate Adaptive Regression Spline (”earth”), Boosted Generalized Linear Model (”glmboost”), Penalized Discriminant Analysis (”pda”), Shrinkage Discriminant Analysis (”sda”), Flexible Discriminant Analysis (”fda”), Lasso and Elastic-Net Regularized Generalized Linear Models (”glmnet”), Penalized Discriminant Analysis (”pda2”), Stepwise Diagonal Linear Discriminant Analysis (”sddaLDA”), Stochastic Gradient Boosting (”gbm”), Multivariate Adaptive Regression Splines (”gcvEarth”), Boosted Logistic Regression (”LogitBoost”), Penalized Multinomial Regression (”multinom”), Penalized Logistic Regression (”plr”), CART (”rpart”), Stepwise Diagonal Quadratic Discriminant Analysis (”sddaQDA”), Generalized Linear Model (”glm”), Nearest Shrunken Centroids (”pam)”, and Cost-Sensitive CART (”rpartCost”)

Super Learner

Super Learner (SL) is a method for selecting an optimal algorithm from a set of candidates using cross-validation. The selection strategy relies on the choice of a loss function (negative log likelihood in the present study) and the choice of a library of candidate algorithms. Comparison of candidate algorithns within a library uses V-fold cross-validation: for each candidate algorithm, SL averages the estimated risks across the validation sets, resulting in the so-called cross-validated risk. Cross-validated risk estimates are used to choose the weighted linear convex combination of the candidate learners with the smallest estimated risk. This convex combination, applied to algorithms run using all of the learning data, is referred to as the SL estimator [21, 26].

We used the Super Learner package in R (Version: 2.0-15) in this study to compare three Super Learner estimators:

SL1 Included only baseline variables with all 23 of the previously identified traditional machine learning algorithms in the SL library.

SL2 Identical to SL1, but with the addition of the hdPS algorithm in its SL library. Note that only the hdPS algorithm had access to the claims code variables in SL2.

SL3 Identical to SL1, but with the addition of claims code covariates selected by the hdPS screening method. Based on the performance of single hdPS algorithms, a fixed pair of hdPS tuning parameters is selected, and SL3 finds the optimal ensemble of all the algorithm candidates fitted on the same baseline and hdPS covariates.

Super Learner	Libray	Covariates
SL1	All machine learning algorithms	Only baseline covariates.
SL2	All machine learning algorithms and the hdPS algorithm	Baseline covariates; Only the hdPS algorithm can use code data.
SL3	All machine learning algorithms	Baseline covariates and hdPS covariates generated from code data by hdPS screening method.

Table 1.1: Details of the three Super Learners considered.

Performance Metrics

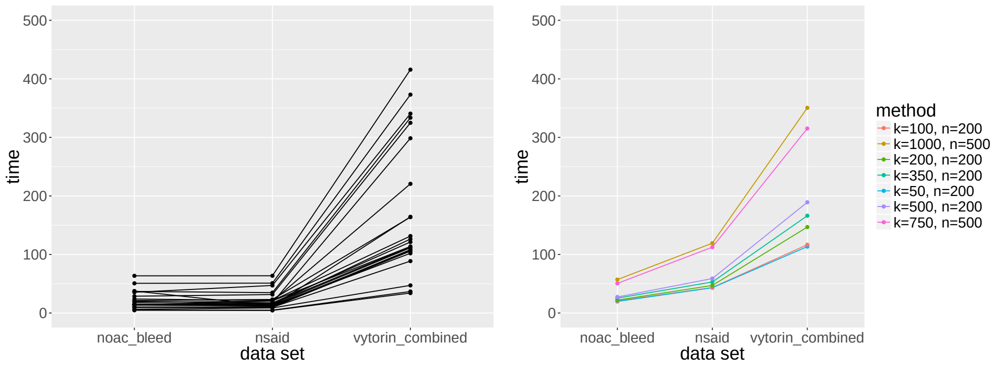
Unlike the causal effect, which is hard to validate, the performance of a prediction model can be validated by the performance on new data. We used three criteria to evaluate our algorithms: computing time, negative log-likelihood, and area under the curve (AUC). In statistics, a receiver operating characteristic (ROC), or ROC curve, is a plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. The curve is created by plotting the true positive rate against the false positive rate at various threshold settings. The AUC is then computed as the area under the ROC curve.

For both computation time and negative log-likelihood, smaller values indicate better performance, whereas for AUC the better classifier achieves greater values [15]. Compared to the error rate, the AUC is a better assesment of performance for the unbalanced classification problem.

Results

Using the hdPS prediction algorithm with Super Learner

Computation Times



(a) Running time for the 23 individual machine learning algorithms with no Super Learner.

(b) Running time for the hdPS algorithms varying the parameter k from 50 to 750 for $n = 200$, and $n = 500$.

Figure 1.1: Running times for individual machine learning and hdPS algorithms without SuperLearner. The y-axis is in log scale.

Figure 1.1 shows the running time for the 23 individual machine learning algorithms and the hdPS algorithm across all three datase without the use of Super Learner. Running time is measured in seconds. Figure 1.1a shows the running time for the machine learning algorithms that only use baseline covariates. Figure 1.1b shows the running time for the hdPS algorithm at varying values of the tuning parameters k and n. The running time is sensitive to n, the number of selected covariates in step 2, while less sensitive to k, the total number of covariates generated by hdPS screening (see details in hdPS section). This suggests most of the running time for hdPS is spent generating and screening covariates. The running time for the hdPS algorithm is generally around the median of all the running time of the machine learning algorithms with only baseline covariates.

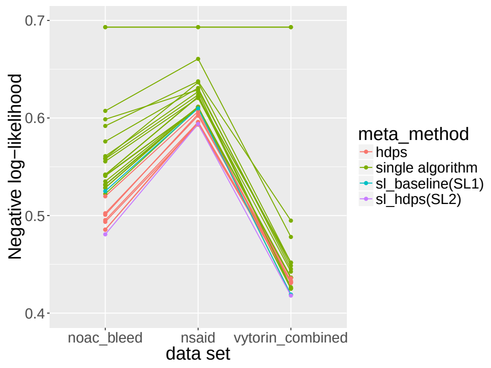
The running time of SL is not placed in the figures. Super Learner with baseline covariates takes just over twice as long as the sum of the running time for each individual algorithms in its library: SL splits data into training and validation set, fits on the training set, finds weights based the on the validation set, and finally retrains the model on the whole set. In other words, Super Learner will fits every single algorithm twice. Therefore, the running time will be about twice the sum of its constituent algorithms, which is what we see in this study (See Table 1.2).

Data Set	Algorithm	Processing Time (seconds)
NOAC	Sum of machine learning algorithms	481.13
	Sum of hdPS algorithms	222.87
	Super Learner 1	1035.43
	Super Learner 2	1636.48
NSAID	Sum of machine learning algorithms	476.09
	Sum of hdPS algorithms	477.32
	Super Learner 1	1101.84
	Super Learner 2	2075.05
VYTORIN	Sum of machine learning algorithms	3982.03
	Sum of hdPS algorithms	1398.01
	Super Learner 1	9165.93
	Super Learner 2	15743.89

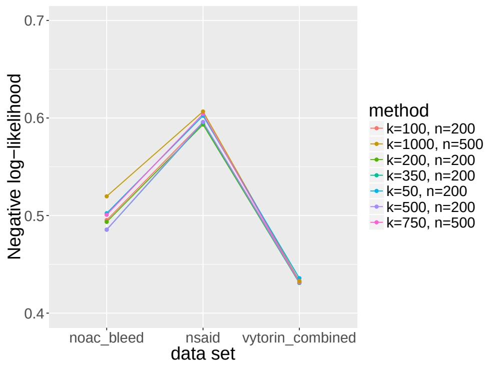
Table 1.2: Running time of the machine learning algorithms, the hdPS algorithms, and Super Learners 1 and 2. Twice the sum of the running time of the machine learning algorithms is comparable to the running time of Super Learner 1 and twice the sum of the running times of both the machine learning algorithms and the hdPS algorithms is comparable to the running time of Super Learner 2.

Super Learner 2 (with a library of all machine learning algorithms and hdPS) takes a bit more than expected. The small extra time is possibly due to SL generating hdps covariates on the validation set based on the fitted hdPS algorithm, which takes a long time if the size of the data is large.

Negative log-likelihood



(a) Negative log-likelihood for SL1, SL2, the hdPS algorithm, and the 23 machine learnng algorithms.



(b) Negative log-likelihood for the hdPS algorithm, varying the parameter k from 50 to 750 for $n = 200$, and $n = 500$.

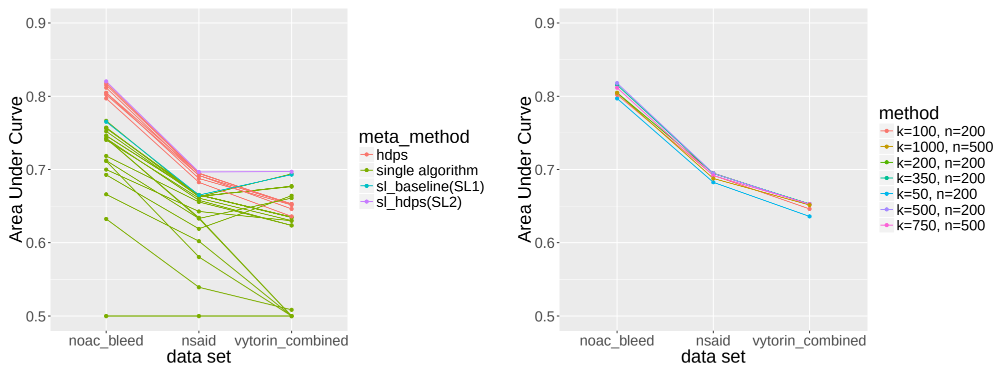
Figure 1.2: The negative log-likelihood for SL1, SL2, the hdPS algorithm, and the 23 machine learning algorithms.

Figure 1.2a shows the negative log-likelihood for Super Learners 1 and 2, and each of the 23 machine learning algorithms (with only baseline covariates) . Figure 1.2b shows the negative log-likelihood for hdPS algorithms with varying tuning parameters, n and k.

The performance of hdPS is not sensitive to either n or k. hdPS outperforms most individual algorithms in the library in most cases, as it takes advantage of the extra information from code data. However, in the Vytorin data set, there are still some machine learning algorithms which perform slightly better than hdPS with respect to the negative log-likelihood.

We can see the SL (without hdPS) outperforms all the other individual algorithms, empirically verifing the optimal property proved by previous literatures [21, 26]: the Super Learner can do at least as well as the best algorithm in the library. The figures show that including the hdPS algorithm improves the performance of Super Learner. With the help of hdPS, Super Learner achieves the best performance among all the algorithms (including hdPS itself). This suggests the time consumption is worthwhile for Super Learner.

AUC



(a) AUC of SL1, SL2, the hdPS algorithm, and the 23 machine learnng algorithms.

(b) AUC for the hdPS algorithm, varying the parameter k from 50 to 750 for $n = 200$, and $n = 500$.

Figure 1.3: The area under the ROC curve (AUC) for for Super Learners 1 and 2, the hdPS algorithm, and each of the 23 machine learning algorithms.

SL uses loss-based cross-validation to select the optimal combination of individual algorithms, so it is not surprising that it outperforms other algorithms with respect to the negative log-likelihood. As propensity score estimation can be considered a binary classification problem, we can use the Area Under the Curve (AUC) to compare performance across algorithms. Binary classification is typically determined by setting a threshold. As the threshold varies for a given classifier we can achieve different true positive rates (TPR) and false positive rates (FPR). A Receiver Operator Curve (ROC) space is defined by FPR and TPR as the x- and y-axes respectively, to depict the trade-off between true positives (benefits) and false positives (costs) at various classification thresholds. We then draw the ROC curve of TPR and FPR for each model and calculate the AUC. The upper bound for a perfect classifier is 1 while a naive random guess would achieve about 0.5.

In Figure 1.3a, we compare the performance of Super Learners 1 and 2, the hdPS algorithm, and each of the 23 machine learning algorithms. Although we optimized the Super Learners with respect to the negative log-likelihood loss function, SL1 and SL2 have outstanding performance with respect to the AUC; Over the NOAC and NSAID data set, SL1 (with only baseline variables) achieves the best AUC compared to all machine learning algorithms in its library, with only a slightly weaker AUC performance than hdPS. In the VYTORIN data set, SL1 outperforms hdPS algorithms with respect to AUC, even though the hdPS algorithms use the additional claims data.

Super Learner 2 clearly combines the strength of the hdPS algorithm and all the machine learning algorithms in its library. Table 1.3 shows, in all three data sets, it achieves higher AUC over all the other algorithms, including hdPS and SL1.

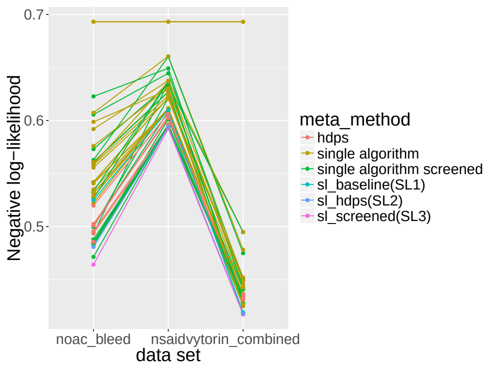
data	SL1	SL2	best hdPS (parameter k/n)
noac	0.7652	0.8203	0.8179 (500/200)
nsaid	0.6651	0.6967	0.6948 (500/200)
vytorin	0.6931	0.6970	0.6527 (750/500)

Table 1.3: Comparison of AUC for SL1, SL2 and best hdPS across three data sets

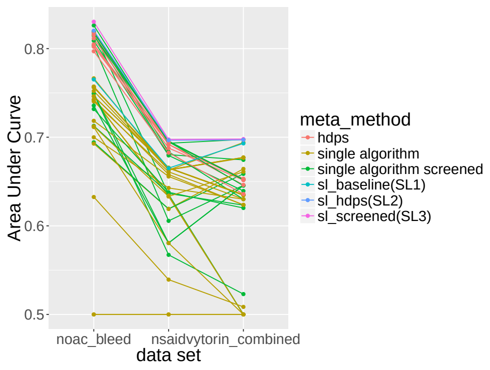
Using the hdPS screening method with Super Learner

In the previous sections, we compared machine learning algorithms limited to only baseline covariates with the hdPS algorithms across different parameters (negative log-likelihood and AUC). The results show that including the hdPS algorithm in a Super Learner library increases performance significantly. In this section, we combine the strength of the claims code data via the hdPS screening method with the machine learning algorithms to improve the propensity score estimation.

We first used the hdPS screening method (with tuning parameters $n = 200, k = 500$) to generate and screen the hdPS covariates. Then we combined these hdPS covariates with the baseline covariates to generate augmented datasets for each of the three datasets under consideration. We build a Super Learner library which included each of the 23 individual machine learning algorithms. Note that, as the original hdPS method uses logistic regression for prediction, it can be considered a special case of LASSO (with $\lambda = 0$).



(a) Negative log-likelihood for SL3 and the sinlge machine learning algorithms.



(b) AUC for SL3 and each of the single machine learning algorithms.

Figure 1.4: Negative log-likelihood and AUC of SL1, SL2, and SL3, compared with each of the single machine learning algorithms (with and without using hdPS covariates). We could see among all the single algorithms and Super Learners, SL3 performs best cross three datasets

For convenience, we differentiate Super Learners 1, 2 and 3 by their algorithm libraries: machine learning algorithms with only baseline covariates, augmenting this library with hdPS, and only the machine learning algorithms but with both baseline and hdPS screened covariates. (See Table 1.1).

Figures 1.4 compares the negative log-likelihood and AUC, respectively, of all three Super Learners and machine learning algorithms. It is clear that the performance of all algorithms increases significantly by including the hdPS screened code covariates. SL3 is slightly better than SL2, and the difference is very small.

Data set	Performance Metric	Super Learner 1	Super Learner2	Super Learner 3
NOAC		0.7652	0.8203	0.8304
NSAID	AUC	0.6651	0.6967	0.6975
VYTORIN		0.6931	0.6970	0.6980
NOAC		0.5251	0.4808	0.4641
NSAID	Negative Log-likelihood	0.6099	0.5939	0.5924
VYTORIN		0.4191	0.4180	0.4171

Table 1.4: Performance as measured by AUC and negative log-likelihood for the three Super Learners with the following libraries: machine learning algorithms with only baseline covariates, augmenting this library with hdPS, and only the machine learning algorithms but with both baseline and hdPS screened covariates. (See Table 1.1).

In table 1.4, we can again see the trend that performance improves from Super Learner 1 to 2 and from 2 to 3. The differences in AUC and in negative log-likelihood between SL1 and 2 are large, while these differences between SL2 and 3 are small. This suggests two things: First, the prediction step in the hdPS algorithm (logistic regression) works well: it performs approximately as well as the best individual machine learning algorithm in the library for Super Learner 3. Second, the hdPS screened covariates make the propensity score estimation more flexible; using SuperLearner we can easily develop different models/algorithms which incorporate the covariate screening method from hdPS.

Weights of Individual Algorithms in Super Learners 1 and 2

Data Set	**Algorithms Selected for SL1**	**Weight**
NOAC	SL.caret.bayesglm_All	0.30
	SL.caret.C5.0_All	0.11
	SL.caret.C5.0Tree_All	0.11
	SL.caret.gbm_All	0.39
	SL.caret.glm_All	0.01
	SL.caret.pda2_All	0.07
	SL.caret.plr_All	0.01
NSAID	SL.caret.C5.0_All	0.06
	SL.caret.C5.0Rules_All	0.01
	SL.caret.C5.0Tree_All	0.06
	SL.caret.ctree2_All	0.01
	SL.caret.gbm_All	0.52
	SL.caret.glm_All	0.35
VYTORIN	SL.caret.gbm_All	0.93
	SL.caret.multinom_All	0.07

Data Set	**Algorithms Selected for SL2**	**Weight**
NOAC	SL.caret.C5.0_screen.baseline	0.03
	SL.caret.C5.0Tree_screen.baseline	0.03
	SL.caret.earth_screen.baseline	0.05
	SL.caret.gcvEarth_screen.baseline	0.05
	SL.caret.pda2_screen.baseline	0.02
	SL.caret.rpart_screen.baseline	0.04
	SL.caret.rpartCost_screen.baseline	0.04
	SL.caret.sddaLDA_screen.baseline	0.03
	SL.caret.sddaQDA_screen.baseline	0.03
	SL.hdps.100_All	0.00
	SL.hdps.350_All	0.48
	SL.hdps.500_All	0.19
NSAID	SL.caret.gbm_screen.baseline	0.24
	SL.caret.sddaLDA_screen.baseline	0.03
	SL.caret.sddaQDA_screen.baseline	0.03
	SL.hdps.100_All	0.25
	SL.hdps.200_All	0.21
	SL.hdps.500_All	0.01
	SL.hdps.1000_All	0.23
VYTORIN	SL.caret.C5.0Rules_screen.baseline	0.01
	SL.caret.gbm_screen.baseline	0.71
	SL.hdps.350_All	0.07
	SL.hdps.750_All	0.04
	SL.hdps.1000_All	0.17

Table 1.5: Non-zero weights of individual algorithms in Super Learners 1 and 2 across all three data sets.

Super Learner produces an optimal ensemble learning algorithm, i.e. a weighted combination of the candidate learners in its library. Table 1.5 shows the weights for all the non-zero weighted algorithms included in the optimal, data set-specific ensemble learner generated by SL 1 and 2. We can see for all the three data sets, the gradient boosting algorithm gbm has the highest weight. It is also interesting to note that across the different data sets the hdPS algorithms have very different weights. Using NOAC and NSAID, the hdPS algorithm plays a dominating role: hdPS algorithms occupy more than 50% of the weight. However in VYTORIN, boosting still plays the most important role, with weight 0.71. This suggests that gradient boosting and hdPS plays a significant role in the prediction of propensity scores. In further studies of prediction/estimation of propensity scores on similar data sets, we may first only include the algorithms with high weights if computation time is limited.

Discussion

Data Set	Method	Negative Log Likelihood	AUC	Negative Log Likelihood (Train)	AUC (Train)	Processing Time (Seconds)
NOAH	k=50, n=200	0.50	0.80	0.51	0.79	19.77
	k=100, n=200	0.50	0.80	0.50	0.80	20.69
	k=200, n=200	0.49	0.80	0.49	0.81	22.02
	k=350, n=200	0.49	0.82	0.47	0.83	25.38
	k=500, n=200	0.49	0.82	0.46	0.84	27.35
	k=750, n=500	0.50	0.81	0.45	0.85	50.58
	k=1000, n=500	0.52	0.80	0.43	0.86	57.08
	sl_baseline	0.53	0.77	0.53	0.77	1035.43
	sl_hdps	0.48	0.82	0.47	0.83	1636.48
NSAID	k=50, n=200	0.60	0.68	0.61	0.67	43.15
	k=100, n=200	0.60	0.69	0.60	0.69	43.48
	k=200, n=200	0.59	0.70	0.60	0.69	47.08
	k=350, n=200	0.60	0.69	0.59	0.70	52.99
	k=500, n=200	0.60	0.69	0.59	0.71	58.90
	k=750, n=500	0.60	0.69	0.58	0.71	112.44
	k=1000, n=500	0.61	0.69	0.58	0.72	119.28
	sl_baseline	0.61	0.67	0.61	0.66	1101.84
	sl_hdps	0.59	0.70	0.59	0.71	2075.05
VYTORIN	k=50, n=200	0.44	0.64	0.43	0.64	113.45
	k=100, n=200	0.43	0.65	0.43	0.65	116.73
	k=200, n=200	0.43	0.65	0.43	0.66	146.81
	k=350, n=200	0.43	0.65	0.42	0.67	166.18
	k=500, n=200	0.43	0.65	0.42	0.67	189.18
	k=750, n=500	0.43	0.65	0.42	0.68	315.22
	k=1000, n=500	0.43	0.65	0.42	0.68	350.45
	sl_baseline	0.42	0.69	0.42	0.70	9165.93
	sl_hdps	0.42	0.70	0.41	0.71	15743.89

Table 1.6: Perfomance for hdPS and SL

Tuning Parameters for hdPS Screening Method

The screening process of hdPS needs to be cross-validated in the same step as its predictive algorithm. For this study, the computation is too expensive for this procedure, so there is an additional risk of overfitting due to the selection of hdPS covariates. A solution would be to generate various hdPS covariate sets under different hdPS hyper parameters and fit the machine learning algorithms on each covariate set. Then, SL3 would find the optimal ensemble among all the hdPS covariate set/learning algorithm combinations.

Performance of hdPS

Although hdPS it is a simple logistic algorithm, it wisely takes advantage of extra information from claims data set. It is therefore reasonable that hdPS outperforms most algorithms in most cases. Processing time for hdPS is sensitive to n while less sensitive of k (see 1.2. The performance is, however, not sensitive to either n or k (see 1.6). Therefore, Super Learners which include hdPS may save processing time by including only a limited selection of hdPS algorithms without sacrificing performance.

Risk of overfitting for hdPS

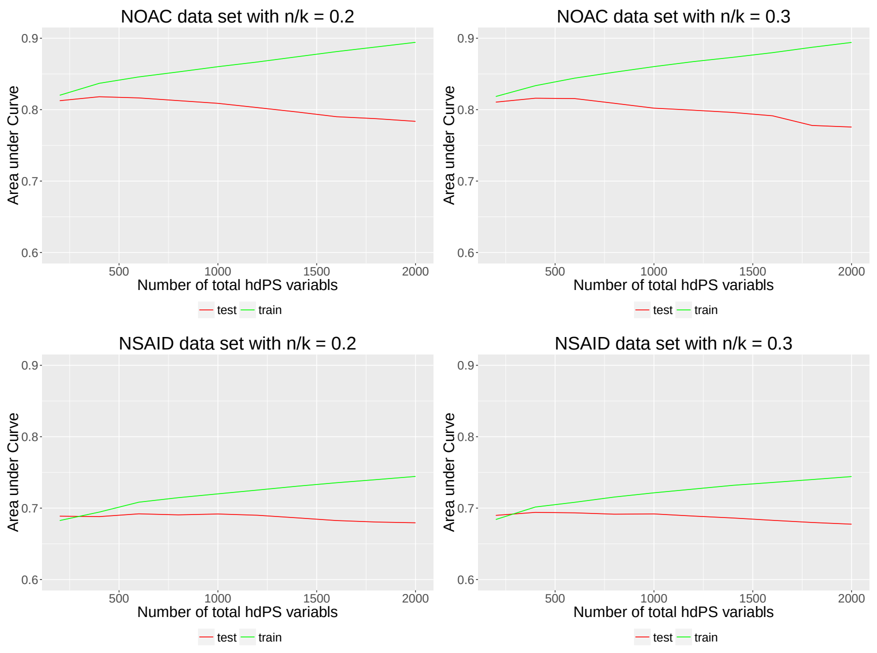


Figure 1.5: AUC for hdPS algorithms with different number of variables, k.

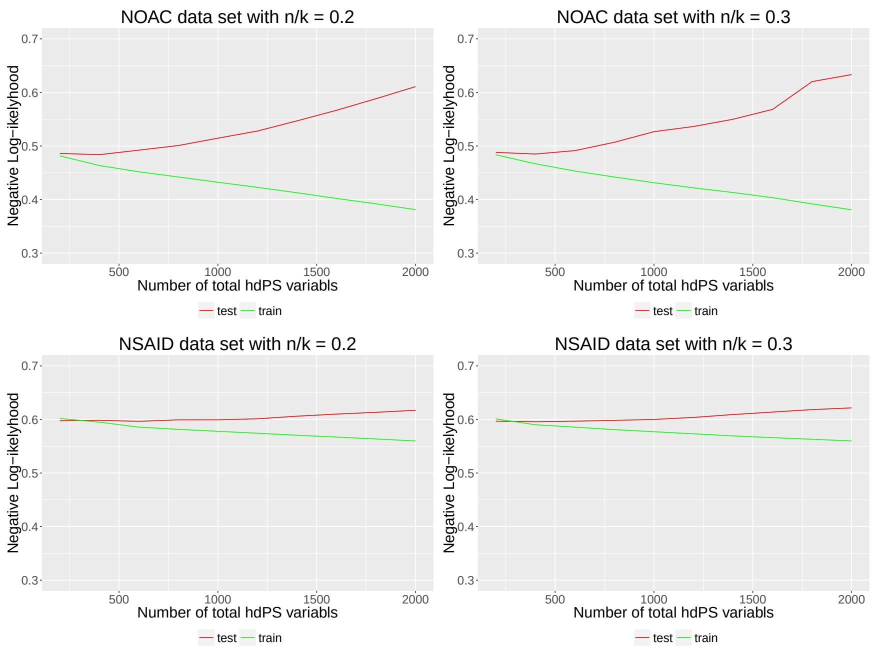


Figure 1.6: Negative loglikelihood for hdPS algorithms with different number of variables, k.

The hdPS algorithm utilizes many more features than traditional methods, which may raise the risk of overfitting. The performance table (table 1.6) shows the negative loglikelihood for both training set and testing set. We can see the difference of performance of hdPS in training set and test set are very small, and not sensitive to k and n. As long as we control k and n in a resonable range, we may not need to worry about the curse of dimensionality.

To study the risk of overfitting for hdPS across each data set, we fix the propotion of number of variable per dimension (n) and number of total hdPS variables (k), then increase k to see the performance of hdPS algorithms. The green lines represent performnce over the training sets and red lines represent peformance over the test sets.

From figure 1.5, we see that increasing the number of variables in hdPS, results in an increase in AUC in the training sets. This is deterministically a result of increasing model complexity. To mitigate this effect, we look at the AUC over the test sets to determine if model complexity reduces performance. For both $n/k = 0.2$ and $n/k = 0.4$, AUC in the testing sets is fairly stable for $k < 500$, but for larger values of k begins to decrease. We find then, that hdPS is only sensitive to overfitting for $k > 500$.

Similarly, in figure 1.6, the negative log-likelihood decreases as k gets larger. The negative log-likelihood in the testing sets begins to increase only for $k > 500$, similary to what we found for AUC. Thus, we conclude the negative log-likelihood is also not sensitive to k for $k < 500$.

Due to the large sample sizes of our datasets, the binary nature of the claims code covariates, and the sparsity of hdPS variables, these hdPS algorithms are at less of a risk of overfitting. However, the high dimensionality data may lead to -some computation issues.

Penalized hdPS

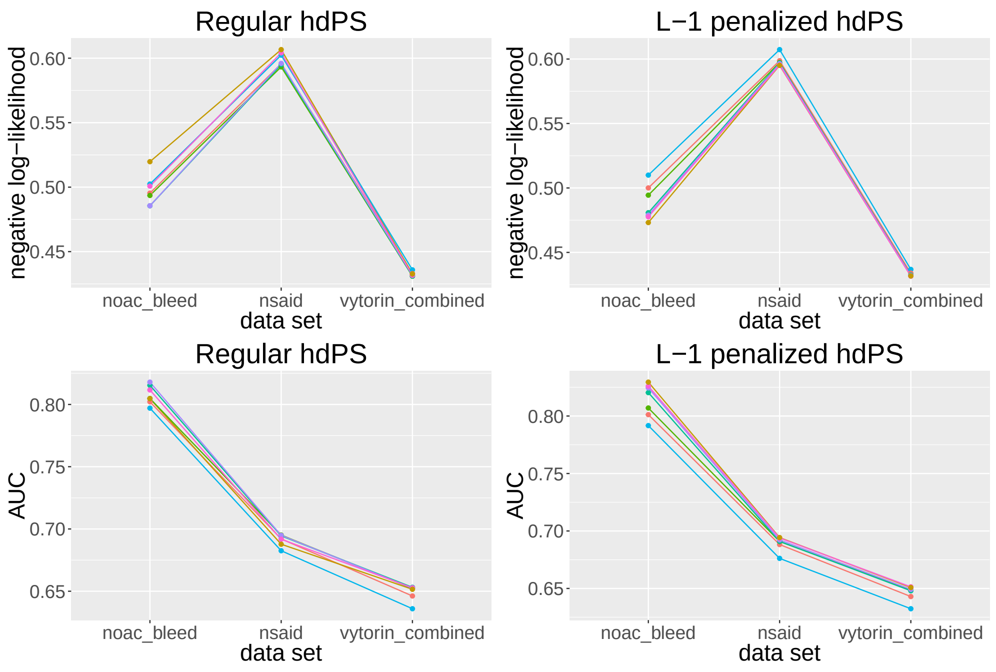


Figure 1.7: Unregularized hdPS Compared with Regularized hdPS

The hdPS algorithm uses multivariate logistic regression for its estimation. We compared the performance of this algorithm against that of regularized regression by implementing the estimation step using the cv.glmnet method in glmnet [12] package in R, which uses cross-validation to find the best tunin parameter λ.

To study if regularization can decrease the risk of overfitting for hdPS, we use $L-1$ regularization (LASSO) for the logistic regression step in hdPS. For every regular hdPS we used cross-validation to find out the best tunning parameter based on discrete Super Learner.

Figure 1.7 shows the negative log-likelihood and AUC over the test sets for unregularized hdPS (left) and regularized hdPS (right). We can see that using regularization can increase performance slightly. In this study, the sample size is relatively large. The regularization does not help a lot. However, when dealing with smaller data set, it is highly suggested to use regularized regression for the last step of hdPS algorithmm, or first generate hdPS covariates and then use Super Learner (as the idea of SL3).

Predictive Performance for SL

SL is a weighted linear combination of candidate learner algorithms that has been demonstrated to perform asymptotically at least as well as the best choice among the library of candidate algorithms, whether or not the library contains a correctly specified parametric statistical model. The results in previous sections show the performance of SL in a finite data set.

From the previous sections, we found that among algorithms which only utilize baseline variables, SL always outperforms the best candidate in its library. Also with respect to the prediction of propensity scores:

- Super Learner reliably outperforms candidate algorithms in AUC, even though the model selection step in SL minimizes a different performance criterion: the cross-validated negative log-likelihoood.

- The hdPS screening method offers a simple way to utilize the information from claims data which increases estimation performance significantly. It is therefore reasonable to take advantage of hdPS covariates in Super Learner.

Data-adaptive property of SL

Besides the outstanding estimation performance, there are several other reasons to use Super Learner for the estimation of propensity scores: First, esimating the propensity score using a parametric model requires accepting strong assumptions concerning the functional form of the relationship between treatment allocation and the covariates, while propensity score model misspecification may result in significant bias in the treatment effect estimate [31, 6]. Second, the relative performance of different algorithms relies heavily on the underlying data generate distribution. Therefore, to avoid model misspecification, we must try many models. This paper clearly demonstrates the strength of this approach: some algorithms perform well over several data sets, but not always. Including many different types of algorithms in the SL library accommodates this inevitability. Cross-validation helps us avoid the risk of over fitting, and so we may include as many algorithms as we can, if the computation consumption permits.

To summarize:

- From figure 1.5, the Gradient Boosting and hdPS have the dominating weight in all three data sets. Hence they are two most powerful individual algorithms for prediction of propensity scores in these three data sets. We may include them first if computation resource is limited.

- The optimal learner for prediction will highly depend on the underlying data-generating distribution: one model may succed in one case, while may fail in another data set. Therefore it is reasonable to use SL including as many competing algorithms as possible.

Conclusion

This article demonstrates the practical performance of Super Learner in finite sample sets for a specific prediction problem (propensity score prediction). The outstanding performances for both negative log-likelihood and AUC demonstrate the reliability of the performance of Super Learner. Based on the cross-validation procedure, SL can adaptively combine a number of different estimators with non-negative weights while avoiding overfitting.

One of the advantages of Super Learner is how easily it can adopt the strengths of field-specific algorithms. The Super Learner framework allows a researcher to try many prediction algorithms, including models based on *a priori* knowledge about the variables, knowing that the final combined Super Learner estimates will either be the best fit or near the best fit [26]. In this paper we successfully implemented SL utilizing the hdPS algorithm to predict propensity scores and achieved optimal performance.

Chapter 2

Online Super Learner for Online Adversarial Prediction

Introduction

Ensemble learning methods train several baseline models, and use some techniques to combine them together to make prediction. The ensemble learning methods have gained popularity because their superior prediction performance. Consider an off-line case with some fixed data generating mechanism. The performance of a particular learner depends on how effective its searching strategy is in approximating the optimal predictor defined by the true data generating distribution [21]. Thus, the relative performance of various learners will depend on the true data-generating distribution, models' assumptions, and their bias variance trade off. However, it is generally impossible to know a priori which learner would perform best given the data and prediction problem. One widely used method is to use cross-validation to give an 'objective' assessment of the learners, and then find the optimal convex combination minimizing the cross-validated risk. [39, 5] proposed a linear combination strategy called stacking to ensemble the models, and [21] proposed cross-validation based optimization framework called Super Learner to determine the best weight for each candidate learner. In this article we extend this off-line cross-validation based ensemble method to the online setting.

General Online Learning and Optimization

In an online convex optimization problem a decision-maker makes a sequence of decisions, i.e., chooses a sequence of points in Euclidean space, from a fixed feasible set. After each point is chosen, it encounters a sequence of (possibly unrelated) convex cost functions [17]. Consider a repeated game with decision method plays $a_t \in \mathcal{A}$, and world reveals loss $l_t \in \mathcal{L}$.

- $\mathcal{A}$: a convex subset of R^d
- $\mathcal{L}$: set of convex real functions on $\mathcal{A}$

At time t:

- Player chooses an action a_t from $\mathcal{A}$
- Adversary chooses $l_t : \mathcal{A} \to R$ from $\mathcal{L}$
- Player incurs loss $l_t(a_t)$

To make the idea of loss more concrete in the online prediction setting, we think we first define a fixed loss function l. Then define $l_t(a) = l(a, y_t)$, which depends on the 'outcome' y_t at round t. In other words, at each round, the adversary select an outcome y. In this study, a_t is the weight for each predictor, with $\mathcal{A}$ be the probability simplex. Consider we have m single algorithms, with n rounds. We define online cumulative loss as:

$$\hat{L}_n = \sum_{i=1}^{n} l_t(a_t)$$

Instead of minimizing the online loss itself, we consider a strategy to minimize the regret:

$$regret = \sum_{t=1}^{n} l_t(a_t) - \sum_{t=1}^{n} l_t(a)$$

In other words, we aim to minimize regret, that is, perform well compared to the best (in retrospect) from some class. For simplicity, we define $L_n^* = \min_{a \in \mathcal{A}} \sum_{t=1}^{n} l_t(a)$.

Notice we do not make any assumptions about the 'data generating mechanism' in online setting. Instead, the adversarial loss (or outcome y_t in the classification setting) could be determined after we make the action a_t, to maximize the regret. Thus we could consider it as an adversarial prediction problem.

Super Learner and Online Super Learner

Stacking and Super Learner

Consider the prediction task in offline setting. The main idea in stacking is to 'stack' the predictions $f_1, \cdots, f_m$ by linear combination with weight a_i:

$$f_{stacking} = \sum_{i=1}^{m} a_i f_i$$

The idea of ensemble learning, which combines predictors instead of selecting single predictors, is well studied in the statistics. [5] summarized and referred many related literatures [27, 9, 32, 3, 14] about the ensemble of predictors. Recently, two widely used ensemble techniques are bagging [4] and boosting [11, 10, 13]. The bagging uses bootstrap aggregation to reduce the variance for the strong learners, while the boosting techniques 'boost' the learning ability for the weak learners.

The idea of stacking originated by [39], which argues the stacking works by deducing the biases of the generalizer(s) with respect to a provided learning set. [5] also studied the stacked regression by using cross-validation to construct the 'good' combination. [21] proposed a general framework called super learner by minimizing cross-validated risk for the combination, and showed its finite sample and asymptotics properties. Literatures show its application to wide range of topics, e.g. survival analysis [18], clinical trial [35], and mortality prediction [25].

For simplicity, we consider the binary classification task, which could be easily generalized to multiclass classification and regression. We first study a simple version of offline super learner with m single classification algorithms, using negative log-likelihood as loss function. It ensemble the base learners by V-fold cross-validation:

$$R_{CV} = \sum_{v=1}^{V} \sum_{i \in val(v)} l(y_i, \mathbf{p}_i^{-v}, a)$$

where $val(v)$ is the index for the observations in the v-th fold, and $\mathbf{p}_i^{-v}$ is defined as the predictions from the i-th base learner that trained with all the data except the v-th fold. There are many options for the loss function. For example, consiger negative log likelihood loss: $l(y_i, \mathbf{p}_i, a) = -(y_i \log(\mathbf{p}_i \cdot a) + (1 - y_i)\log(1 - \mathbf{p}_i \cdot a))$, with constrain a on a probability simplex:

$$||a||_1 = 1, a_i \geq 0, \text{for } i = 1, \cdots, m$$

Online Super Learner

Consider an online adversarial classification problem (e.g. spam detection): Consider a repeated game: every round world first reveals some input covariates X_t, m baseline algorithms gives predition of outcome $\mathbf{p_t} = (p_{t1}, \cdots, p_{tm})$, online Super Learner computes the weight for ensemble: $a_t \in \mathcal{A}$, and world reveals the outcome y_t. At time t:

- There are m base learners (they could be either pre-trained algorithm, or online algorithm like online SVM and online boosting). The i-th algorithm gives the prediction p_{ti} for the probability for outcome to be 1, based on the input covariates X_t.
- Players choose some strategy to give a estimated probability based on the predicted probability $\mathbf{p}_t$ from m base learners by convex combination with weight a_t. See more details in the later sections.
- Adversary chooses the true outcome, y_t, and therefore the current online loss incurred is $l_t = l(y_t, a_t, \mathbf{p}_t) : \mathcal{A} \to R$ from $\mathcal{L}$, where l could be 0-1 loss, or some surrogate loss.
- Player incurs loss $l_t(a_t)$

Online Super Learner with negative log likelihood loss could be considered as an online convex optimization problem. Consider at each rount t, we have m predictors $p_{t,1}, \cdots p_{t,m}$ (could be considered as the prediction from m online classifiaction algorithm, or experts). We want to find a weight on a probalisitc simplex ($|a|_1 = 1$ with $a_i \geq 0$) for each prediction algorithms to minimize the regret:

$$regret = \sum_{i=1}^{n}(l(\mathbf{a_t}, \mathbf{p_t}, y_t)) - \sum_{i=1}^{n}(l(\mathbf{a}, \mathbf{p_t}, y_t))$$

where $l(\mathbf{a}, \mathbf{p_t}, y_t)$ is a loss function. Aforementioned, we focus on the binary classification, and $l(\mathbf{a}, \mathbf{p_t}, y_t)$ is the negative log likelihood for Super Learner prediction:

$$l(\mathbf{a}, \mathbf{p_t}, y_t) = y_t \log(\mathbf{a} \cdot \mathbf{p_t}) + (1 - y_t)\log(1 - \mathbf{a} \cdot \mathbf{p_t})$$

It could be solved by online regularized convex optimization using mirror descent. Also the constrain of a could be achieved by Bregman projection or Lazy projection at each step. The bounded domain for a and bounded gradient of loss function guarantee the online regret could be upper bounded with order $\mathcal{O}(\sqrt{n})$ w.r.t. the number of iterations.

Other Notations

We start by establishing the notation used throughout the paper. In this paper, we focus on the binary classification. Without any specification, we assume $y \in \{0, 1\}$ and $\tilde{y} \in \{-1, 1\}$. We use bold face letters (e.g. $\mathbf{a}$) to define a vector, and standard lower case letters to define a scalar (e.g. y). The dot product of bold characters are the dot product of two vectors: $\mathbf{a} \cdot \mathbf{p} = \sum_{i=1}^{m} a_i p_i$. If with no extra specification, $||\cdot||$ stands for the Euclidean norm $||\cdot||_2$.

$\nabla f(a)$ is the gradient for the function f at point a, while $\partial f(a)$ denotes the subgradient for the function f at point a.

We define logit function as $logit(t) = \ln(\frac{p}{1-p})$ and expit function as $expit(t) = logit^{-1}(t) = \frac{exp(t)}{1+exp(t)}$.

Simple Linear Super Learner with Negative Log-Likelihood Loss

Solve Linear Online Super Learner by Mirror Descent

One potential way to solve online super learner is update the weight $\mathbf{a}$ by minimizing the sum of previous loss:

$$a_{t+1} = \arg\min_{a \in A}(\sum_{s=1}^{t} l(\mathbf{a}, \mathbf{p_s}, y_s))$$

This method uses the idea of 'follow the leader'. It could be useful if the loss function have good property (e.g. strongly convex), or we have strong assumption of the single learner and the data streaming generating system. However, in online setting, it has been proved that the adversary could choose adversarial data that makes 'follow the leader' strategy fail (with order $\mathcal{O}(n)$ regret).

To solve the original online convex optimization, consider a regularized minimization approach:

$$a_{t+1} = \arg\min_{a \in A}(\eta \sum_{s=1}^{t} l(\mathbf{a}, \mathbf{p_s}, y_s) + R(a))$$

where R is some regularization to make the sequence stable. Different regularization leads to different online procedure. The solution to the regularized optimization could be computed by mirror descent. In this study, we define $R(a)$ as $l-2$ regularization $\frac{1}{2}||a||_2^2$. To minimize

This is a typical constrained online convex optimization problem. We could solve this by mirror descent:

Algorithm 1 Online Mirror Descent for Convex Loss

```
1: Define the link function $g(\theta) = \arg\min_{a \in \mathcal{A}}(\eta\langle a, \theta\rangle + R(a))$
2: Initialize $a = (\frac{1}{m}, \cdots, \frac{1}{m})$
3: for each round $t$ do
4:     Predict with $a_t = g(\theta_t)$
5:     Update $\theta_{t+1} = \theta_t + z_t$, where $z_t \in \partial f_t(w_t)$
6: end for
```

As we take $l-2$ regularization in this case, we have $g(\theta) = \eta \cdot \theta$. Thus in unconstrained case with $l-2$ regularization, we could just take online gradient descent:

$$\tilde{a}_{t+1} = \tilde{a}_t - \eta g_t$$

Then we consider the constrain: we want a lies on the probability simplex, which means we constrain a with $||a||_1 = 1$ and $a_i \geq 0$. This constrain could be achieved by Bregman projection:

Theorem 1. *[34]*

For constrained constrained minimization is equivalent to unconstrained minimization, followed by Bregman projection

$$a_{t+1} = \arg\min_{a \in \mathcal{A}} \Phi_t(a)$$

$$\tilde{a}_{t+1} = \arg\min_{a \in R^m} \Phi_t(a)$$

then the constrained solution equals to the Bregman projection of the unconstrained solution:

$$a_{t+1} = \Pi_{\mathcal{A}}^{\Phi_t} \tilde{a}_{t+1}$$

Thus we could solve the constrained regularized online convex optimization by online gradient descent and then take Bregman projection at each step. For $l-2$ regularization and negative log-likelihood loss, the projection is:

$$\Pi_{\mathcal{A}}^{\Phi_t}(\tilde{a}_{t+1}) = \arg\min_{a \in \mathcal{A}} \Phi_t(a) - (\Phi_t(b) + \nabla \Phi_t(b)(a-b))$$

As the loss function is nonlinear, the projection would be complex and slow. We could sidestep this by 'lazy projection': we use $l-2$ projection instead of the Bregman projection with respect to the loss. Euclidean projection on $l-1$ ball could solved efficiently by fast algorithm proposed by [7] in $\mathcal{O}(m)$ expected time. In addition, we would show by lazy projection, we could still achieve good regret bound.

Here is the pseudo code for the online super learner with negative log likelihood loss:

Algorithm 2 Online Linear Super Learner solved by Gradient Descent

```
1: for each round $t$ do
2:     Compute the current gradient: $\nabla l_t(a) = y_t \frac{\mathbf{p_t}}{\mathbf{a \cdot p_t}} + (1-y_t)(\frac{-\mathbf{p_t}}{1-\mathbf{ap_t}})$
3:     Update combination weight $\tilde{a}_{t+1} - \eta \nabla l_t(a)$
4:     $l-2$ project $\tilde{a}_{t+1}$ onto the $l-1$ unit ball: $a_{t+1} = \Pi_{\mathcal{A}} \tilde{a}_{t+1}$
5: end for
```

Theorem 2. *The regret of this online super learner solved by online gradient descent is upper bounded by*

$$\hat{L}_n - L_n^* \leq \frac{2\sqrt{nm}}{\gamma}$$

Proof. To prove the theorem 2, we first introduce the following theorem:

Theorem 3. *[40] For $G = \max_t ||\nabla l_t(a_t)||$ and $D = diam(A)$, the gradient strategy with $\eta = D/(G\sqrt{n})$ and lazy projection has regret satisfying:*

$$\hat{L}_n - L_n^* \leq GD\sqrt{n}$$

We want to bound the Euclidean norm of the gradient as well as the diameter of $\mathcal{A}$. First as $\mathcal{A} = \{||a||_1 = 1, a_i \geq 0, \text{for } i = 1, \cdots, m\}$, for any $a \in \mathcal{A}$, we have $||a||_2 \leq 1$. Thus $D = 2$.

To give an upper bound for the regret, we need extra assumption to bound $p_{t,i}$

Assumption 1. *Positivity: We assume all the predicted probability from the online classifiers are bounded awary from 0 and 1. In other words, for any input X, the predicted probability would be within $[\gamma, 1-\gamma]$ for some small constant γ.*

Notice this might not true, but in practice we could trim the predicted probability by $[\gamma, 1-\gamma]$ to make the gradient within a reasonable range. Thus the Euclidean norm of gradient is bounded by:

With such assumption the gradient at round t is $\nabla l_t(a_t) = y_t \frac{\mathbf{p_t}}{\mathbf{a}\cdot\mathbf{p_t}} + (1-y_t)(\frac{-\mathbf{p_t}}{1-\mathbf{a}\cdot\mathbf{p_t}})$, we have $\max_t ||\nabla l_t(a_t)||_2 \leq ||(\frac{1}{\gamma}, \cdots, \frac{1}{\gamma})||_2 = \frac{\sqrt{m}}{\gamma}$.

Take $D=2$, $G=\sqrt{m}$ into Theorem 3, we get the upper bound for the regret for the log likelihood loss is $\frac{2\sqrt{nm}}{\gamma}$.

$\square$

Solve Linear Online Super Learner by Second Order Method with Logarithmic Regret

Hazan et al.[17] proposed several methods that achieve logarithmic regret bound. To achieve logarithmic bound for online gradient descent, we need strong assumption for the loss function: l_t must be H-strong convex. This is not easy to achieve in online prediction. They also proposed an online Newton method, which only requires a mild condition that the loss function is α-exp-concave.

Definition 1. *If there is an $\alpha > 0$ such that $\exp(-\alpha l_t(a))$ is a concave function for $a \in \mathcal{A}$, for all t:*

$$\forall a \in \mathcal{A}, t \in [n], \nabla^2[\exp(-\alpha l_t(a))] \prec 0$$

We call such class of cost functions satisfy the α-exp-concavity.

Algorithm 3 Online Newton Method [17]

```
1: Use any arbitrary point $a_i \in \mathcal{A}$
2: Define $\beta = \frac{1}{2}\min(\frac{1}{4GD}, \alpha)$ and $\epsilon = \frac{1}{B^2D^2}$ , $\nabla_t = \nabla l_t(a_t)$and $A_t = \sum_{i=1}^{t} \nabla_t \nabla_t^T + \epsilon I_n$
3: for each round $t$ do
4:     $\tilde{a}_{t+1} = x_{t-1} - \frac{1}{\beta} A_{t1}^{-1} \nabla_{t1}$
5:     Project $\tilde{a}_{t+1}$ onto $\mathcal{A}$: $a_{t+1} = \Pi_{\mathcal{A}}^{A_t-1} \tilde{a}_{t+1}$
6:     where $\Pi_{\mathcal{A}}^{A_t-1}$ is defined by $\Pi_{\mathcal{A}}^{A_t-1}(\tilde{a}) = \arg\min_{a\in\mathcal{A}}(a-\tilde{a})A_{t-1}(a-\tilde{a})$
7: end for
```

Theorem 4. *[17] Assume that for all t, the loss function l_t is α-exp-concave, with $diam(\mathcal{A}) \leq D$ and $||\nabla l(x)|| \leq G$, the algorithm Online Newton Step has the following bound:*

$$regret \leq 5(\frac{1}{\alpha} + GD)m\log(n)$$

For simplicity, we assume $\tilde{y} = 2y - 1$. We could write the negative log likelihood loss for the linear super learner as the logistic loss: $l_t(\mathbf{a}) = \log(1 + e^{-\tilde{y}_t(\mathbf{p_t}\cdot\mathbf{a} - \frac{1}{2})})$. It achieves minimum when $\tilde{y}_t(2\mathbf{p_t}\cdot\mathbf{a}-1)=1$, which means $\mathbf{y_t}\cdot a$ matches $\tilde{y}_t$ perfectly. It can be easily verified that

$$\begin{aligned}\nabla l_t(\mathbf{a}) &= \frac{-1}{1+e^{\tilde{y}_t(\mathbf{p_t}\mathbf{a}-\frac{1}{2})}}\tilde{y}_t\mathbf{p_t}\\ \nabla^2 l_t(\mathbf{a}) &= \frac{1}{2+e^{\tilde{y}_t(\mathbf{p_t}\mathbf{a}-\frac{1}{2})}+e^{-\tilde{y}_t(\mathbf{p_t}\mathbf{a}-\frac{1}{2})}}\mathbf{p_t}\cdot\mathbf{p_t}^T\end{aligned}$$

Thus, $G := sup||\nabla l_t(\mathbf{a})||_2 \le \frac{\sqrt{m}}{1+\frac{1}{\sqrt{2}}}$. And $\nabla^2 l_t(\mathbf{a})$ is a positive semi-definitive matrix, which implies that $l_t(\mathbf{a})$ is a convex function. Moreover, suppose $g_t(\mathbf{a}) = exp(-\alpha l_t) = (1+e^{-\tilde{y}_t(\mathbf{p_t}\cdot\mathbf{a}-\frac{1}{2})})^{-\alpha}$. A few calculations lead to

$$\begin{aligned}\nabla g(\mathbf{a}) &= \alpha(1+e^{-\tilde{y}_t(\mathbf{p_t}\cdot\mathbf{a}-\frac{1}{2})})^{-(\alpha+1)}\cdot e^{-\tilde{y}_t(\mathbf{p_t}\cdot\mathbf{a}-\frac{1}{2})}\tilde{y}_t\mathbf{p_t}\\ \nabla^2 g(\mathbf{a}) &= \left(\frac{\alpha+1}{1+e^{\tilde{y}_t(\mathbf{p_t}\cdot\mathbf{a}-\frac{1}{2})}}-1\right)\alpha(1+e^{-\tilde{y}_t(\mathbf{p_t}\cdot\mathbf{a}-\frac{1}{2})})^{-(\alpha+1)}e^{-\tilde{y}_t(\mathbf{p_t}\cdot\mathbf{a}-\frac{1}{2})}\mathbf{p_t}\cdot\mathbf{p_t}^T\end{aligned}$$

$g(\mathbf{a})$ is a concave function $\Leftrightarrow \nabla^2 g(\mathbf{a})$ is a negative semi-definitive matrix $\Leftrightarrow \left(\frac{\alpha+1}{1+e^{\tilde{y}_t(\mathbf{p_t}\cdot\mathbf{a}-\frac{1}{2})}}-1\right) \le 0$. It can be guaranteed when $\alpha \le \frac{1}{\sqrt{e}}$. To apply this loss function to Algorithm 3, according to [17], we set can $G = \frac{\sqrt{m}}{1+\frac{1}{\sqrt{2}}}$, $D=\sqrt{2}$, $\alpha=\frac{1}{\sqrt{e}}$. Thus we have:

$$regret \le 5(\sqrt{e}+\sqrt{2}\frac{\sqrt{m}}{1+\frac{1}{\sqrt{2}}})m\log(n)$$

Non-linear Extension of the Simple Super Learner: Deeper Ensemble

We showed simple super learner with the predicted probabilities from single algorithms could be solved in online mode by online mirror descent or online Newton method, with non-trivial regret upper bounded. However, this in practice does not outperform much compared to other ensemble method like online bagging, or exponential weight strategy (multiple expert advice). We could improve this algorithm by 'stacking' the algorithms in deeper level: We first use *logit* function to transform the original predicted probabilities, then find a 'good' convex combination, and then apply *expit* transformation to $[0,1]$ scale. In the batch mode, the combination weight could be computed based on cross-validation:

$$R_{CV}(\mathbf{a}) = \sum_{v=1}^{V}\sum_{i\in val(v)}\ln(y\cdot expit(logit(\mathbf{p}_i^v)\mathbf{a})+(1-y)(1-expit(logit(\mathbf{p}_i^v)\mathbf{a})))$$

Consider a transformed outcome $\tilde{y}\in\{-1,1\}$ by $\tilde{y}=2y-1$, this equals the loss for logistic regression:

$$R_{CV}(\mathbf{a}) = \sum_{v=1}^{V} \sum_{i \in val(v)} \ln(1 + exp(-\tilde{y}_i(logit(\mathbf{p})\mathbf{a})))$$

Easy to check this loss is convex with respect to the weight **a**. For the online mode, the loss is no longer linear, thus we could not use regularized online optimization with mirror descent to solve it. However, as it is convex, we could still achieve an upper bound for the regret under some assumption.

Define $z_{t+1} = \tilde{y}_{t+1} logit(\mathbf{p_{t+1} a_t})$, then the gradient of the loss is:

$$\frac{\partial l_{t+1}}{\partial a_i} = -\frac{exp(-z_{t+1})}{1 + exp(-z_{t+1})} \cdot \tilde{y} \cdot logit(p_{t+1}[i])$$

Due to Assumption 1 for the predicted probability, we have:

$$||\nabla l_t|| \leq -\sqrt{m} \cdot logit(\gamma)$$

Here we do not have restriction of the weight **a**, but to achieve the theoretical upper bound for the regret, we still could impose restriction on $\mathcal{A}$. For example, we could restrict $\mathcal{A}$ be a non-negative Euclidean ball with diameter D. Take these into theorem 3, we could get the upper bound for this deeper non-linear super learner:

$$regret \leq -logit(\gamma) D \sqrt{nm}$$

However, this bound could be very large if γ is small.

Here is the pseudo code for the extension of simple online super learner:

Algorithm 4 Online Non-Linear Super Learner solved by Gradient Descent

```
1: for each round $t$ do
2:     Compute the current gradient: $\nabla l_t(a) = -\frac{exp(-z_t)}{1+exp(-z_t)} \cdot \tilde{y} \cdot logit(p_i)$
3:     Update combination weight $\tilde{a}_{t+1} - \nabla l_t(a)$
4:     $l-2$ project $\tilde{a}_{t+1}$ onto $\mathcal{A}$: $a_{t+1} = \Pi_{\mathcal{A}} \tilde{a}_{t+1}$
5: end for
```

Experiments

Other Competitors

Exponential Weight Strategy

The first version of online super learner is simple and interpretable. Howeber, it does not work well on the data set. In comparison, the second (linear and solved by online Newton method) and third version of super learner (non-linear and solved by gradient descent) perform much better in our empirical studies. In this section, we compare the two versions of online super learner with several well-known competitors.

First consider exponential weight strategy:

Algorithm 5 Exponential weight strategy

```
1: Consider each single algorithm is a expert with prediction $e_{t,i}$ at round $t$
2: Choose $a_t$ as the normalized vector $\frac{w_t}{||w_t||_1}$ and
3: Maintain a set of unnormalized weight over experts: $w_1^i = 1, w_{t+1}^i = w_t^i exp(-\eta l_t(e_i))$,
   where $\eta$ is a parameter of the algorithm and $l_t(e_i)$ is loss incurred by expert $i$
4: Choosing the prediction from one expert randomly, according to the distribution $a_t$
```

Theorem 5. *[34] The exponential weights strategy with parameter*

$$\eta = \sqrt{\frac{8\ln(m)}{n}}$$

has regret satisfying

$$\hat{L}_n - L_n^* \leq \sqrt{\frac{n\ln(m)}{2}}$$

In this study, we use two kinds of loss for the online prediction: the first is the negative log likelihood, and the second is 0-1 loss. Though the regrets for both of them could be bounded by order $\mathcal{O}(\sqrt{n})$, the loss functions are different. We will study the difference of the performance in later section.

Online Bagging

We also compare the online super learner to the online bagging algorithm [23]. The idea of online bagging is to mimic bootstrap by generating Poisson random variables:

Algorithm 6 Online Bagging

```
1: for Each round $t$ do
2:     Compute the current predictions from single algorithms, and take average of them
       as bagging prediction
3:     for Each $i$-th single algorithm do
4:         Generate a Poisson random variable $k_{t,i}$
5:         Train the $i$-th single online algorithm $k_{t,i}$ times
6:     end for
7: end for
```

Though there is no theoretical result about the bound for the regret about the online bagging, empirical studies shows it works well in online prediction.

Limitation of the Experiment

The number online classification algorithms is limited compared with off-line prediction algorithms. For similicity, we only consider the online linear models with different parametric formula solved by gradient descent as the single algorithm. Then we assess the performance of the each ensemble method as well as the best parametric model in each case.

Ensemble Models Description

In the later sections, we refer 'super learner 1' as the algorithm 4, 'super learner 2' as the linear super learner with logistic loss solved by online Newton steps (algorithm 3). We slightly modify exponential weight with cross-entropy loss by solving it use Newton method instead of gradient and we call it 2rd order exp weight. We refer online bagging as the algorithm 6. 'exp weight 1' and 'exp weight 2' are the algorithm 5 with cross-entropy loss and 0-1 loss, respectively.

Simulation

In the simulation study, we consider four simulation cases to mimic the real world data generation mechanism. In the first two simulation, data are generated i.i.d. from a fixed data generating system.

For simulation 1, we have

$$P(Y_t = 1) = expit(1 + X_1 - 2X_2 + X_3 - 2.5X_4)$$

And each parametric learners are main term logistic regression on two of the covariates. This mimic the senario that each base learner only have the partial information. Intuitively, the linear model with X_2, X_4 will be in favor as the coefficient for them are larger.

For simulation 2, we have:

$$P(Y_t = 1) = expit(1 + X_1 - 2X_2 + X_3 - 2.5X_4 + 2\sin(X_1))$$

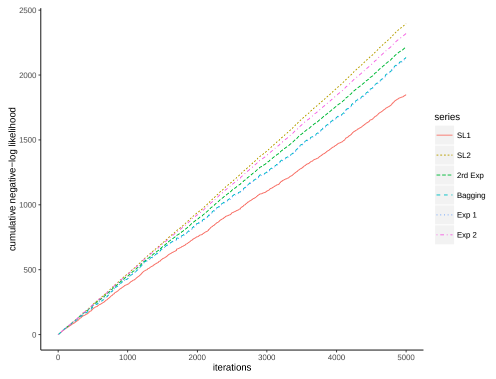
The base learners are 1. main term for all covariates. 2. second order interaction for all covariates. 3. Second order interaction for all pairs of three covariates. In this case, models are flexible, but true function is not in the model space.

For simulation 3 and 4, we consider there is a change point in the data streaming. In other words,

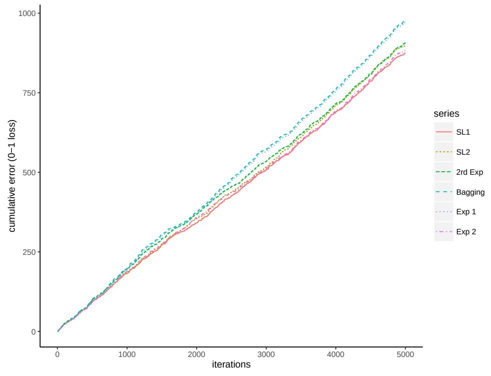
$$P(Y_t = 1) = expit(1 + X_1 - 2X_2 + X_3) \text{ for } t = 1, \cdots, 2500$$
$$P(Y_t = 1) = expit(1 - 2X_2 + X_3 - 2.5X_4) \text{ for } t = 2501, \cdots, 5000$$

The base learner are slightly different. For simulation 3, we use the same idea as simulation 1 by setting all the base learners with two covariates. For simulation 4, we consider candidates with more complex parametric formula.

For all the simulations, the sample size for online data stream is 5000. Also we use 500 extra data as 'historical' data to initialize each single learner.

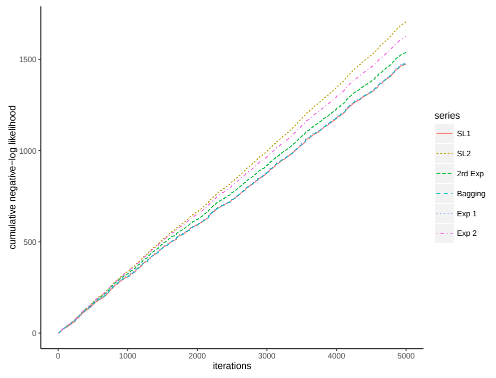


(a) Online Cumulative Negative Log Likelihood

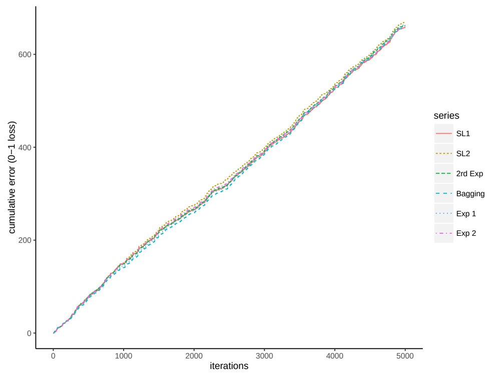


(b) Online Cumulative Error

Figure 2.1: Online cross entropy loss and error for simulation 1



(a) Online Cumulative Negative Log Likelihood



(b) Online Cumulative Error

Figure 2.2: Online cross entropy loss and error for simulation 2

Figure 2.1 and 2.2 show the online loss (negative log likelihood) and error for each ensemble algorithms. With extra power from nonlinearity, super learner 1 outperforms other algorithms significantly with respect to the negative log likelihood. Though it still have best performance with respect to error, the difference is small. This might suggest in practice, outstanding performance w.r.t. the surrogate loss is a good criterion, but not always guarantee the error rate.

Table 2.1: Cross-entropy and error rate for 6 ensemble methods and the best single algorithm for 4 simulations

Simulation	Loss	super learner 1	super learner 2	2rd order exp weight	bagging	exp weight 1	exp weight 2	Best Baseline
1	Cross-entropy	1818.964	2395.625	2265.373	2319.478	2121.994	2121.286	2131.095
	Error rate	0.173	0.18	0.1766	0.1766	0.1952	0.1938	0.196
2	Cross-entropy	1474.773	1706.039	1538.267	1625.915	1477.748	1482.17	1482.018
	Error rate	0.1316	0.1340	0.1324	0.1318	0.1318	0.1328	0.1332
3	Cross-entropy	2461.748	2777.04	2660.266	2711.715	2523.297	2554.554	2557.132
	Error rate	0.2466	0.279	0.2578	0.2638	0.2454	0.2472	0.2554
4	Cross-entropy	2513.693	2557.248	2589.645	2572.772	2618.957	2775.929	2614.527
	Error rate	0.2568	0.2546	0.2532	0.2542	0.2454	0.2544	0.2492

Table 2.1 shows the online risk for 6 ensemble methods. As we run all the model with same data, L^* would be same. Thus we could directly compare online risk $\hat{L}_n$ in stead of regret. Online bagging shows good performance in simulation 4 while bad performance in 3. This is because of the property of the base learner: in simulation 4, base learners are strong learners, thus use averaging could reduce the variance. However, when the learners are weak, averaging models would not make much difference. This suggest data adaptive weight for the models is necessary in online setting.

We could see the nonlinear super learner achieves satisfactory performance. Though the difference for the error rate is small, the cross entropy loss is smaller than other algorithms across all simulations. This might suggest the nonlinear transformation for the input p_t is necessary.

Data Description

We use several data sets from UCI machine learning repository [22] and Knowledge Extraction based on Evolutionary Learning (KEEL) data repository [2, 1] and LIBSVM data set [19]. Due to the limitation of the base algorithm, we select the data sets with number of features from 2 to 20.

Table 2.2: Sample size and number of features for each dataset

data	Total size	Number of features
svmguide1	7089	4
banana	5300	2
titanic	2201	3
magic	19020	10
phoneme	5404	5
ring	7400	20

We compare the performance of the ensemble methods from two perspectives: first we compare the online loss for each ensemble method. We also split 20% of each data set as testing set, and make prediction on the testing set. We record the online and offline performance for each ensemble method.

Table 2.3: Error rate for online and offline prediction for all the ensemble methods, as well as the best single algorithm

data		super learner 1	super learner 2	2rd order exp weight	online bagging	exp weight 1	exp weight 2	Best Baseline
svmguide1	online error rate	0.0612	0.0672	0.0595	0.0599	0.0643	0.0665	0.0630
	offline testing error rate	0.0439	0.0464	0.0450	0.0406	0.0561	0.0567	0.0558
banana	online error rate	0.462	0.471	0.472	0.470	0.472	0.461	0.450
	offline testing error rate	0.448	0.462	0.466	0.463	0.467	0.448	0.449
titanic	online error rate	0.229	0.223	0.225	0.226	0.227	0.221	0.209
	offline testing error rate	0.224	0.216	0.217	0.217	0.217	0.209	0.209
magic	online error rate	0.294	0.307	0.312	0.323	0.311	0.311	0.310
	offline testing error rate	0.322	0.256	0.272	0.347	0.268	0.267	0.251
phoneme	online error rate	0.231	0.245	0.244	0.245	0.243	0.247	0.231
	offline testing error rate	0.226	0.242	0.239	0.241	0.229	0.238	0.227
ring	online error rate	0.335	0.345	0.339	0.339	0.507	0.350	0.350
	offline testing error rate	0.329	0.367	0.322	0.322	0.501	0.334	0.339

Table 2.3 shows the prediction error (0-1 error) for both online and offline prediction from each online ensemble method across all datasets.

Discussion and Future Works

From table 2.3, we could see the performance of super learner 1 is satisfactory, no matter in online or offline setting. It is slightly better compared with super learner 2, which uses linear combination of predictor and online Newton steps. This might because the linear combination is too simple to construct a complex hypothesis from base learners. Further more, the online Newton steps might focus on adversarial case and thus too conservative in general prediction setting.

In addition, we notice in the ring data set, the error for exp weight 1 is significantly large. This might because the regret is optimized with respect to the negative log likelihood: even if this regret is small, the regret for error might be large.

Due to the limitation of time, we do not offer details of the comparison of nonlinear Super Learner solved by Newton method and linear super learner solved by gradient descent. More comparison of the different level of ensemble and solving strategy need to be studied. In addition, the base learner in the library for online super learner is limited in this study. Different algorithm candidates might influence the performance of the ensemble method. Also more adversarial data examples could be studied. We leave this as the future work.

Conclusion

In this study, we proposed an online version of super learner. It could be solved by online gradient descent, or online Newton steps, with $\mathcal{O}(\sqrt{n})$ and $\mathcal{O}(\log(n))$ upper bound for regret, respectively. For online Newton steps, we proposed a new online projection algorithm to accelerate the projection step. Then we develop a deeper super learner by ensemble all the predictors in the logit level. The performance of the online super learner algorithms is assessed on the four simulations and 6 real data sets. We also discuss the limitation of the experiment and leave some future work.

Bibliography

[1] J Alcalá et al. “Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework”. In: *Journal of Multiple-Valued Logic and Soft Computing* 17.255-287 (2010), p. 11.

[2] Jesus Alcala-Fdez et al. “KEEL: a software tool to assess evolutionary algorithms for data mining problems”. In: *Soft Computing* 13.3 (2009), pp. 307–318.

[3] James O Berger and ME Bock. “Combining independent normal mean estimation problems with unknown variances”. In: *The Annals of Statistics* (1976), pp. 642–648.

[4] Leo Breiman. “Bagging predictors”. In: *Machine learning* 24.2 (1996), pp. 123–140.

[5] Leo Breiman. “Stacked regressions”. In: *Machine learning* 24.1 (1996), pp. 49–64.

[6] M Alan Brookhart et al. “Variable selection for propensity score models”. In: *American journal of epidemiology* 163.12 (2006), pp. 1149–1156.

[7] John Duchi et al. “Efficient projections onto the l 1-ball for learning in high dimensions”. In: *Proceedings of the 25th international conference on Machine learning.* ACM. 2008, pp. 272–279.

[8] Sandrine Dudoit and Mark J van der Laan. “Asymptotics of cross-validated risk estimation in estimator selection and performance assessment”. In: *Statistical Methodology* 2.2 (2005), pp. 131–154.

[9] Bradley Efron and Carl Morris. “Combining possibly related estimation problems”. In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1973), pp. 379–421.

[10] Yoav Freund and Robert E Schapire. “A decision-theoretic generalization of on-line learning and an application to boosting”. In: *Journal of computer and system sciences* 55.1 (1997), pp. 119–139.

[11] Yoav Freund, Robert E Schapire, et al. “Experiments with a new boosting algorithm”. In: *ICML*. Vol. 96. 1996, pp. 148–156.

[12] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. “glmnet: Lasso and elastic-net regularized generalized linear models”. In: *R package version* 1 (2009).

[13] Jerome H Friedman. “Greedy function approximation: a gradient boosting machine”. In: *Annals of statistics* (2001), pp. 1189–1232.

[14] Edwin J Green and William E Strawderman. "A James-Stein type estimator for combining unbiased and possibly biased estimators". In: *Journal of the American Statistical Association* 86.416 (1991), pp. 1001–1006.

[15] James A Hanley and Barbara J McNeil. "The meaning and use of the area under a receiver operating characteristic (ROC) curve." In: *Radiology* 143.1 (1982), pp. 29–36.

[16] Trevor Hastie et al. *The elements of statistical learning*. Vol. 2. Springer, 2009.

[17] Elad Hazan, Amit Agarwal, and Satyen Kale. "Logarithmic regret algorithms for online convex optimization". In: *Machine Learning* 69.2-3 (2007), pp. 169–192.

[18] Torsten Hothorn et al. "Survival ensembles". In: *Biostatistics* 7.3 (2006), pp. 355–373.

[19] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. "A practical guide to support vector classification". In: (2003).

[20] Max Kuhn et al. "caret: Classification and regression training". In: *R package version* 2 (2012).

[21] 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).

[22] M. Lichman. *UCI Machine Learning Repository*. 2013. URL: `http://archive.ics.uci.edu/ml`.

[23] Nikunj C Oza. "Online bagging and boosting". In: *Systems, man and cybernetics, 2005 IEEE international conference on*. Vol. 3. IEEE. 2005, pp. 2340–2345.

[24] Romain Pirracchio, Maya L Petersen, and Mark van der Laan. "Improving Propensity Score Estimators' Robustness to Model Misspecification Using Super Learner". In: *American journal of epidemiology* 181.2 (2015), pp. 108–119.

[25] Romain Pirracchio et al. "Mortality prediction in intensive care units with the Super ICU Learner Algorithm (SICULA): a population-based study". In: *The Lancet Respiratory Medicine* 3.1 (2015), pp. 42–52.

[26] Eric C Polley and Mark J van der Laan. "Super learner in prediction". In: (2010).

[27] JNK Rao and Kathleen Subrahmaniam. "Combining independent estimators and estimation in linear regression with unequal variances". In: *Biometrics* (1971), pp. 971–990.

[28] Jeremy A Rassen and Sebastian Schneeweiss. "Using high-dimensional propensity scores to automate confounding control in a distributed medical product safety surveillance system". In: *Pharmacoepidemiology and drug safety* 21.S1 (2012), pp. 41–49.

[29] Jeremy A Rassen et al. "Covariate selection in high-dimensional propensity score analyses of treatment effects in small samples". In: *American journal of epidemiology* 173.12 (2011), pp. 1404–1413.

[30] Paul R Rosenbaum and Donald B Rubin. "The central role of the propensity score in observational studies for causal effects". In: *Biometrika* 70.1 (1983), pp. 41–55.

[31] Donald B Rubin. “On principles for modeling propensity scores in medical research”. In: *Pharmacoepidemiology and drug safety* 13.12 (2004), pp. 855–857.

[32] Donald B Rubin and Sanford Weisberg. “The variance of a linear combination of independent estimators using estimated weights”. In: *Biometrika* 62.3 (1975), pp. 708–709.

[33] Sebastian Schneeweiss et al. “High-dimensional propensity score adjustment in studies of treatment effects using health care claims data”. In: *Epidemiology (Cambridge, Mass.)* 20.4 (2009), p. 512.

[34] Shai Shalev-Shwartz. “Online learning and online convex optimization”. In: *Foundations and Trends in Machine Learning* 4.2 (2011), pp. 107–194.

[35] Sandra E Sinisi et al. “Super learning: an application to the prediction of HIV-1 drug resistance”. In: *Statistical applications in genetics and molecular biology* 6.1 (2007).

[36] Sengwee Toh, Luis A García Rodríguez, and Miguel A Hernán. “Confounding adjustment via a semi-automated high-dimensional propensity score algorithm: an application to electronic medical records”. In: *Pharmacoepidemiology and drug safety* 20.8 (2011), pp. 849–857.

[37] Aad W Vaart, Sandrine Dudoit, and Mark J Laan. “Oracle inequalities for multi-fold cross validation”. In: *Statistics & Decisions* 24.3 (2006), pp. 351–371.

[38] Mark J Van Der Laan and Sandrine Dudoit. “Unified cross-validation methodology for selection among estimators and a general cross-validated adaptive epsilon-net estimator: Finite sample oracle inequalities and examples”. In: (2003).

[39] David H Wolpert. “Stacked generalization”. In: *Neural networks* 5.2 (1992), pp. 241–259.

[40] Martin Zinkevich. “Online convex programming and generalized infinitesimal gradient ascent”. In: (2003).