University of California, Berkeley
U.C. Berkeley Division of Biostatistics Working Paper Series

Year 2013 *Paper* 313

Subsemble: An Ensemble Method for Combining Subset-Specific Algorithm Fits

Stephanie Sapp* Mark J. van der Laan†
John Canny‡

*University of California - Berkeley, sapp@berkeley.edu
†University of California - Berkeley, laan@berkeley.edu
‡University of California, Berkeley, jfc@cs.berkeley.edu

Subsemble: An Ensemble Method for Combining Subset-Specific Algorithm Fits

Stephanie Sapp, Mark J. van der Laan, and John Canny

Abstract

Ensemble methods using the same underlying algorithm trained on different subsets of observations have recently received increased attention as practical prediction tools for massive datasets. We propose Subsemble: a general subset ensemble prediction method, which can be used for small, moderate, or large datasets. Subsemble partitions the full dataset into subsets of observations, fits a specified underlying algorithm on each subset, and uses a clever form of V-fold cross-validation to output a prediction function that combines the subset-specific fits. We give an oracle result that provides a theoretical performance guarantee for Subsemble. Through simulations, we demonstrate that Subsemble can be a beneficial tool for small to moderate sized datasets, and often has better prediction performance than the underlying algorithm fit just once on the full dataset. We also describe how to include Subsemble as a candidate in a SuperLearner library, providing a practical way to evaluate the performance of Subsemble relative to the underlying algorithm fit just once on the full dataset.

1 Introduction

As massive datasets become increasingly common, new scalable approaches to prediction are needed. Recently, there has been increased interest in the performance of various subsetting prediction procedures. Subsetting procedures obtain subsets of the full available dataset, train the same underlying algorithm on each subset, and finally combine the results across the subsets. The method used to obtain the subsets, and the method used to combine the subset-specific results, differ depending on the procedure. Prediction methods using subsets of the full available dataset are promising tools for large-scale datasets, since computation on subsets can be parallelized, taking advantage of modern computational resources.

Bagging, developed in Breiman (1996a), is a classic example of a subsampling prediction procedure. Bagging, or bootstrap aggregating, involves drawing many bootstrap samples of a fixed size, fitting the same underlying algorithm on each bootstrap sample, and obtaining the final prediction by simply averaging the results across the subset fits. This approach has several drawbacks. First, some observations will never be used, while others will be selected multiple times. Second, taking a simple average of the subset fits does not differentiate between the quality of each fit.

An average mixture (AVGM) procedure for fitting the parameter of a parametric model has been studied by Zhang et al. (2012). AVGM partitions the full available dataset into disjoint subsets, estimates the parameter within each subset, and finally combines the estimates by simple averaging. Zhang et al. (2012) also propose a bootstrap average mixture (BAVGM) procedure, which extends AVGM. As with AVGM, BAVGM partitions the full data, and estimates the parameter within each subset. However, BAVGM also takes a single bootstrap sample from each partition, re-estimates the parameter on the bootstrap sample, and combines the two estimates into a so-called bootstrap bias corrected estimate. The final parameter estimate is obtained by simple averaging of the bootstrap bias-corrected estimates from each partition. The AVGM and BAVGM procedures have shortcomings. The approaches are only designed for parametric models, and the theoretical results provided rely on using parametric models. AVGM does not account for fit quality differences at all, since it simple averages the subset fits. BAVGM's approach to bias correction estimates the bias of a partition's parameter estimate by reusing data that was already used in the fit of that parameter. Finally, both methods are only proposed for use with large datasets. That is, the methods are proposed due to their computational attractiveness, rather than their statistical performance.

An ensemble method for classification with large-scale datasets, using subsets of observations to train algorithms, and combining the classifiers linearly, was discussed in the case study of Lin and Kolcz (2012). Although Lin and Kolcz (2012) mention the possibility of weighting each classifier if different underlying algorithms are used, they indicate that simple averaging is preferred when using different subsets of observations to train the same underlying algorithm. As their work is a case study, no theoretical performance guarantees are provided. Furthermore, the approach is only evaluated for a single algorithm (logistic regression), with a single dataset, using very large subsets. Finally, the method is again only proposed by the authors for use with large datasets.

We propose a novel method, Subsemble, for combining results from fitting the same underlying algorithm on different subsets of observations. Our approach has many benefits and differs from existing methods in a variety of ways. Any type of underlying algorithm, parametric or nonparametric, can be used. Instead of simply averaging subset-specific fits, Subsemble differentiates fit quality across the subsets and learns a weighted combination of the subset-specific fits. To evaluate fit quality and determine the weighted combination, Subsemble uses cross-validation, thus using independent data to train and learn the weighted combination. Finally, Subsemble has desirable statistical performance and can improve prediction quality on both small and large datasets.

This paper focuses on the statistical performance of Subsemble. We provide an oracle result for Subsemble, showing that Subsemble performs as well as the best possible combination of the subset-specific fits. We describe how to choose between Subsemble and the underlying algorithm fit just once on the full dataset, resulting in a weighted combination of the procedures. Through simulation studies, we demonstrate the desirable performance of Subsemble as a prediction procedure for moderate sized datasets. We show that Subsemble often provides better prediction performance than fitting the underlying algorithm only once on the full available dataset, and that including both the usual and Subsemble versions of algorithms in a SuperLearner library provides superior results to including only the usual versions of algorithms.

The remainder of our paper is organized as follows. Subsemble is presented in Section 2. We describe how to choose between fitting an algorithm just once on the full dataset versus various Subsemble fits, through including both the Subsemble and usual versions of the algorithm as candidates in a SuperLearner library, in Section 3. Simulation study and real data analysis results appear in Section 4. We conclude and discuss future research directions in Section 5.

2 Subsemble

2.1 The Subsemble Algorithm

Assume the full dataset consists of n independent and identically distributed observations $O_i = (X_i, Y_i)$ of $O \sim P_0$. Our goal is to predict the outcome Y_i given the covariate vector X_i. Given an algorithm $\hat{\Psi}$, which is a mapping from an empirical probability distribution P_n into the parameter space space Ψ of functions of X, the usual approach to prediction using $\hat{\Psi}$ applies $\hat{\Psi}$ to the empirical distribution P_n, resulting in the estimator $\hat{\Psi}(P_n)$.

The Subsemble procedure takes a different approach to forming a prediction function using $\hat{\Psi}$. Instead of using the entire dataset to obtain a single fit of $\hat{\Psi}$, Subsemble applies $\hat{\Psi}$ to multiple empirical distributions, each consisting of a subset of the available observations, created from a partitioning of the entire dataset into J disjoint subsets. We refer to these J subsets of the entire dataset at the *final subsets*. Subsemble then obtains the optimal combination of the final subset-

specific fits by minimizing cross-validated risk through V-fold cross-validation.

Note that the cross-validation within Subsemble is used as an estimator selection tool. It is used to find the best combination of subset-specific fits by minimizing cross-validated risk. Risk estimates are based on obtaining subset-specific fits on cross-validation training sets, and estimating risk using the corresponding test sets. For this procedure to yield accurate risk estimates, the j^{th} subset-specific estimator in the cross-validation training sets needs to be similar to the final j^{th} subset-specific estimator of the full dataset. Otherwise, the risk estimate of the j^{th} estimator does not reflect its true risk, and the resulting combination of the J estimators is also meaningless.

The j^{th} estimator is defined as applying the underlying algorithm $\hat{\Psi}$ to the j^{th} final subset. In fact, the only difference between the J estimators is the particular data used to train the underlying algorithm. We thus need to define the j^{th} estimator in the cross-validation training sets to be very similar to the j^{th} final estimator. This is accomplished by using very similar data in the j^{th} cross-validation and final subsets.

To motivate the construction of the V folds used in Subsemble, consider randomly splitting the entire dataset into V folds. Now, suppose that at each cross-validation step, the training data were randomly assigned to the J subsets. With this approach, the data used in subset j in a cross-validation training set has no relationship to the data used in the final subset j. A partial solution would be, at each cross-validation step, to assign the training data to subsets based on each observation's assignment in the final subsets. This construction guarantees that each observation used in the subset-specific fit j during cross-validation is contained in the data used in the final subset-specific fit j. However, undefined estimates could occur if all data in the final subset j happened to fall in the same fold v.

Subsemble instead selects the V folds to preserve the subset structure: we first partition each subset j into V folds, and then create the overall v^{th} fold by combining the v^{th} folds from all the J subsets. This approach has several benefits. First, very similar data is used in the cross-validation subset assignments and the final subset assignments. Second, since only $1/V$ of each final subset is left out at each cross-validation step, the potential problem of undefined estimates in the cross-validation steps is avoided. Finally, creating the cross-validation training sets does not require combining data across the subsets. This is due to the fact that, since the final subsets are partitioned into V folds, and the subset assignments in the cross-validation steps are the same as the final subset assignments, leaving a fold v out of subset j produces all the data assigned to the j^{th} subset in the cross-validation training set. See Figure 1 for an illustration.

Subsemble also requires specifying a second algorithm $\hat{\Phi}$ to be used for combining the subset-specific fits. For example, the combination algorithm $\hat{\Phi}$ could be a linear regression, random forest, or support vector machine. See Figure 1 for an illustration of the Subsemble procedure when $\hat{\Phi}$ is specified as linear regression.

More formally, Subsemble proceeds as follows. Given the user-specified number of subsets J, the

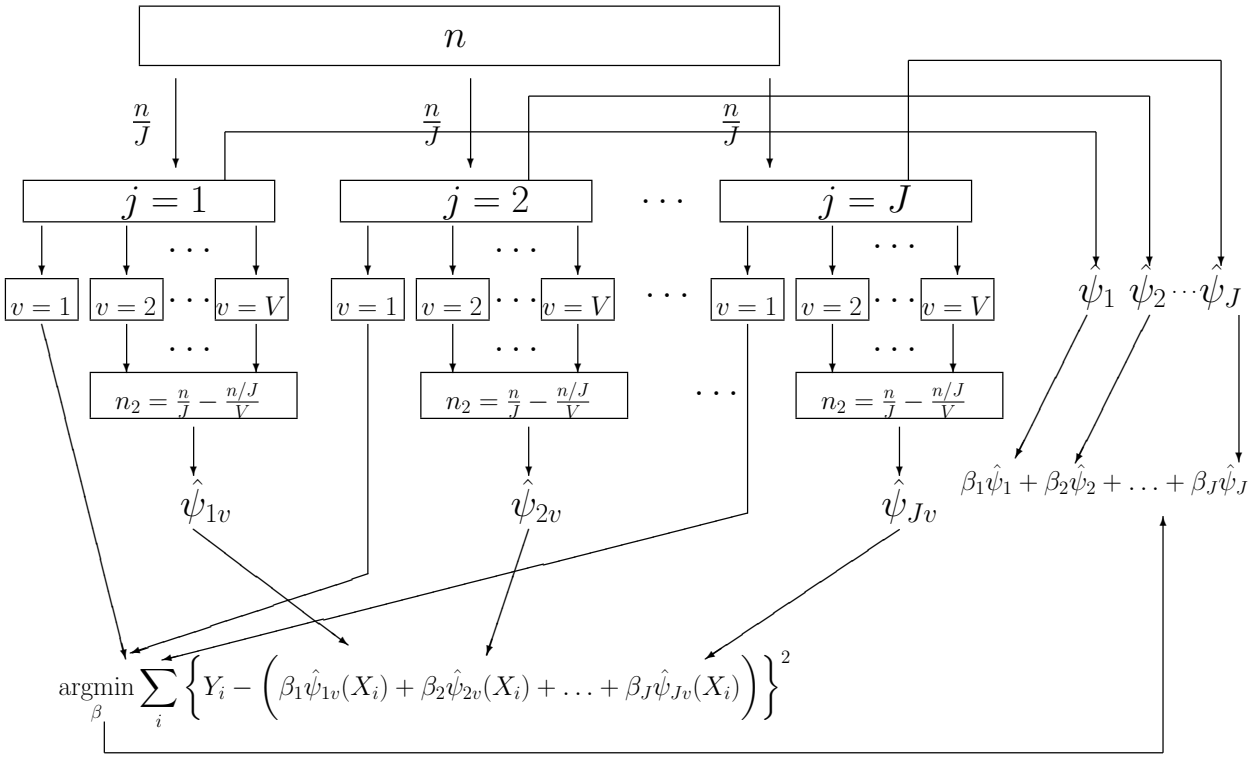

Figure 1: *Diagram of the Subsemble procedure using linear regression to combine the subset-specific fits. The full dataset, consisting of n observations is partitioned into J disjoint subsets. The same underlying algorithm $\hat{\psi}$ is applied to each subset, resulting in J subset-specific fits $\hat{\psi}_1, \hat{\psi}_2, \ldots, \hat{\psi}_J$. V-fold cross-validation, where the V folds are constructed to preserve the subset structure, is used to learn the best weighted linear combination of the subset-specific fits.*

n observations are partitioned into *J* disjoint subsets. Define the algorithm $\hat{\Psi}_j$ as $\hat{\Psi}$ applied to the j^{th} subset. Each of the *J* algorithms $\hat{\Psi}_j$ are applied to P_n, resulting in *J* subset-specific estimators $\hat{\Psi}_j(P_n)$. V-fold cross-validation is then used to select the optimal combination of the subset-specific fits based on minimizing the cross-validated risk. The *V* folds are selected as follows. Each subset $j = 1, \ldots, J$ is first partitioned into V folds. Each full fold *v* is then obtained by combining the v^{th} folds across the *J* subsets. Define $P_{n,v}$ as the empirical distribution of the observations not in the v^{th} fold. For each observation *i*, define $P_{n,v(i)}$ to be the empirical distribution of the observations not in the fold containing observation *i*. The optimal combination is selected by applying the combination algorithm $\hat{\Phi}$ to the following redefined set of *n* observations: $(\tilde{X}_i, Y_i)$, where $\tilde{X}_i = \{\hat{\Psi}_j(P_{n,v(i)})(X_i)\}_{j=1}^J$. That is, for each *i*, the redefined input vector $\tilde{X}_i$ consists of

the J predicted values obtained by evaluating the J subset-specific estimators trained on the data excluding the $v(i)^{\text{th}}$ fold, at X_i. As an example, specifying $\hat{\Phi}$ as linear regression would result in selecting the best linear combination $\sum_{j=1}^{J} \beta_j \hat{\Psi}_j$ of the subset-specific fits, by regressing Y_i onto the J values of $\hat{\Psi}_j(P_{n,v(i)})(X_i)$.

2.2 Oracle Result for Subsemble

The following oracle result, following directly from the work of van der Laan et al. (2007), gives a theoretical guarantee of Subsemble's performance.

Theorem 1. *Assume the combination algorithm $\hat{\Phi} = \hat{\Phi}_\beta$ is indexed by a finite dimensional parameter $\beta \in \mathbf{B}$. Let $\mathbf{B}_n$ be a finite set of values in $\mathbf{B}$, with the number of values growing at most polynomial rate in n. Assume there exist bounded sets $\mathbf{Y} \in \mathbb{R}$ and Euclidean $\mathbf{X}$ such that $P((Y,X) \in \mathbf{Y} \times \mathbf{X}) = 1$ and $P(\hat{\Psi}(P_n) \in \mathbf{Y}) = 1$.*

Define the cross-validation selector of β as

$$\beta_n = \arg\min_{\beta \in \mathbf{B}_n} \sum_{i=1}^{n} \left\{ Y_i - \hat{\Phi}_\beta(\tilde{X}_i) \right\}^2$$

and define the oracle selector of β as

$$\tilde{\beta}_n = \arg\min_{\beta \in \mathbf{B}_n} \frac{1}{V} \sum_{v=1}^{V} E_0 \left[\left\{ E_0[Y|X] - \hat{\Phi}_\beta(P_{n,v}) \right\}^2 \right]$$

Then, for every $\delta > 0$, there exists a constant $C(\delta) < \infty$ (defined in van der Laan et al. *(2006)) such that*

$$E \frac{1}{V} \sum_{v=1}^{V} E_0 \left[\left\{ E_0[Y|X] - \hat{\Phi}_{\beta_n}(P_{n,v}) \right\}^2 \right] \leq (1+\delta) E \frac{1}{V} \sum_{v=1}^{V} E_0 \left[\left\{ E_0[Y|X] - \hat{\Phi}_{\tilde{\beta}_n}(P_{n,v}) \right\}^2 \right] + C(\delta) \frac{V \log n}{n}$$

As a result, if none of the subset-specific learners converge at a parametric rate, then the oracle selector does not converge at a parametric rate, and the cross-validation estimator $\hat{\Phi}_{\beta_n}$ is asymptotically equivalent with the oracle estimator $\hat{\Phi}_{\tilde{\beta}_n}$. Otherwise, the cross-validation estimator $\hat{\Phi}_{\beta_n}$ achieves a near parametric $\frac{\log n}{n}$ rate.

Theorem 1 tells us that the risk difference, based on squared-error loss, of the Subsemble from the true $E_0[Y|X]$ can be bounded from above by a function of the risk difference of the oracle procedure. Note that the oracle procedure results in the best possible combination of the subset-specific fits, since the oracle procedure selects β to minimize the *true* risk difference. As a result, the

main lesson from this Theorem is, since usually the underlying algorithm used wont convergence at parametric rate, Subsemble performs as well as the best possible combination of subset-specific fits.

Note that Theorem 1 doesn't tell us how many subsets are best, or how Subsemble's combination of many subset-specific fits will perform relative to fitting the single algorithm $\hat{\Psi}$ just once on the full available dataset. In Section 3, we provide a practical way to select between Subsemble and a single fit on the full dataset. We also show through simulations in Section 4 that there is often a range of subsets which are better than the full fit.

3 Deciding When to Use Subsembles

3.1 Including Subsembles as Candidates in SuperLearner

While the oracle result for Subsemble given in Section 2.2 provides a theoretical basis for the performance of Subsemble, it doesn't tell us whether or not Subsemble will outperform the standard single fit of an algorithm only once on the entire dataset. The oracle result also provides no guidance about the best number of partitions to use in Subsemble. Here, we provide a practical approach to select between these options, describing how to include Subsembles with different numbers of subsets, as well as the usual version of the specified algorithm, as candidate algorithms in a SuperLearner library.

SuperLearner, developed in van der Laan et al. (2007), is a powerful prediction algorithm that finds the optimal weighted combination of a set of candidate prediction algorithms by minimizing cross-validated risk. SuperLearner generalizes stacking algorithms developed by Wolpert (1992) and extended by Breiman (1996b), and was named based on the theoretical performance results discussed in van der Laan et al. (2007).

The SuperLearner algorithm proceeds as follows. Propose a library of K candidate prediction algorithms. Split the dataset into V blocks of equal size. For each block $v = 1, \ldots, V$, fit each of the K candidate algorithms on the observations not in the v^{th} block, and obtain K predictions for each observation in the v^{th} block using these fits. Select the optimal combination by applying the user-specified minimum cross-validated risk predictor algorithm $\hat{\Theta}$: regressing the true outcome of the n observations on the K predictions to obtain a combination of the K algorithms. Finally, fit the K algorithms on the complete dataset. Predictions are then obtained by using these final fits combined as specified by $\hat{\Theta}$ obtained in the previous step. For additional details, we refer the reader to van der Laan et al. (2007).

SuperLearner can be used to evaluate between Subsembles using different number of subsets, and underlying algorithms fit just once on entire dataset. Simply include Subsembles as candidates in

SuperLearner library, and the underlying algorithms fit once on all data as another candidate. The SuperLearner will then learn the optimal weighted combination of these candidates.

3.2 Oracle Result for SuperLearner

The SuperLearner algorithm has its own oracle result. As developed in van der Laan et al. (2007), we have the following Theorem.

Theorem 2. *Assume the minimum cross-validated risk predictor algorithm $\hat{\Theta} = \hat{\Theta}_\alpha$ is indexed by a finite dimensional parameter $\alpha \in \mathbf{A}$. Let K be the total number of algorithms included in the SuperLearner library, including both full and Subsemble versions. Let $\mathbf{A}_n$ be a finite set of values in $\mathbf{A}$, with the number of values growing at most polynomial rate in n. Assume there exist bounded sets $\mathbf{Y} \in \mathbb{R}$ and Euclidean $\mathbf{X}$ such that $P((Y,X) \in \mathbf{Y} \times \mathbf{X}) = 1$ and $P(\hat{\Psi}_k(P_n) \in \mathbf{Y}) = 1$.*

Define the cross-validation selector of α as

$$\alpha_n = \arg\min_{\alpha \in \mathbf{A}_n} \sum_{i=1}^{n} \left\{ Y_i - \hat{\Theta}_\alpha(\tilde{X}_i) \right\}^2$$

and define the oracle selector of α as

$$\tilde{\alpha}_n = \arg\min_{\alpha \in \mathbf{A}_n} \frac{1}{V} \sum_{v=1}^{V} E_0 \left[\left\{ E_0[Y|X] - \hat{\Theta}_\alpha(P_{n,v}) \right\}^2 \right]$$

Then, for every $\delta > 0$, there exists a constant $C(\delta) < \infty$ (defined in van der Laan et al. *(2006)) such that*

$$E \frac{1}{V} \sum_{v=1}^{V} E_0 \left[\left\{ E_0[Y|X] - \hat{\Theta}_{\alpha_n}(P_{n,v}) \right\}^2 \right] \leq (1+\delta) E \frac{1}{V} \sum_{v=1}^{V} E_0 \left[\left\{ E_0[Y|X] - \hat{\Theta}_{\tilde{\alpha}_n}(P_{n,v}) \right\}^2 \right] + C(\delta) \frac{V \log n}{n}$$

As a result, if none of the learners included in the library converge at a parametric rate, then the oracle selector does not converge at a parametric rate, and the cross-validation estimator $\hat{\Theta}_{\alpha_n}$ is asymptotically equivalent with the oracle estimator $\hat{\Theta}_{\tilde{\alpha}_n}$. Otherwise, the cross-validation estimator $\hat{\Theta}_{\alpha_n}$ achieves a near parametric $\frac{\log n}{n}$ rate.

Similar to the oracle result for Subsemble, Theorem 2 tells us that the risk difference, based on squared-error loss, of the SuperLearner from the true $E_0[Y|X]$ can be bounded from above by a function of the risk difference of the oracle procedure. The oracle procedure results in the best possible combination of the candidate algorithms, since the oracle procedure chooses α to minimize the *true* risk difference. Typically, none of the candidate algorithms will converge at a parametric rate. As a result, SuperLearner will perform as well as best possible combination of candidates.

4 Data Analysis

4.1 Description of Datasets

In the studies that follow, we used four datasets (Sim 1, Sim 2, Yacht, Diamond) to evaluate the practical performance of Subsemble. All datasets have one real-valued output variable, and no missing values.

The first two datasets are simulated, and generated as below. The Sim 1 dataset has 20 input variables. The sim 2 dataset has 200 input variables.

Sim 1:

$$\begin{aligned} X_i &\sim N(0,9),\ i = 1,\ldots,20 \\ \varepsilon &\sim N(0,9) \\ Y &= \varepsilon + X_1 + \sin(X_2) + \log(|X_3|) + X_4^2 + X_5 X_6 + I(X_7 X_8 X_9 < 0) + I(X_{10} > 0) \\ &\quad + X_{11} I(X_{11} > 0) + \sqrt{|X_{12}|} + \cos(X_{13}) + 2X_{14} + |X_{15}| + I(X_{16} < -1) \\ &\quad + X_{17} I(X_{17} < -1) - 2X_{18} - X_{19} X_{20} \end{aligned}$$

Sim 2:

$$\begin{aligned} X_i &\sim N(0,16),\ i = 1,\ldots,200 \\ \varepsilon &\sim N(0,25) \\ Y &= -1 + \varepsilon + \sum_{i=1}^{200} \log(|X_i|) \end{aligned}$$

The second two datasets are publicly available real-world data. The yacht dataset, available from Bache and Lichman (2013), has 308 observations and 6 input variables. The diamond dataset, described by Chu (2001), has 308 observations and 17 input variables.

4.2 Subsemble Performance Comparison

In this study, we compare the performance of Subsemble with two alternatives: fitting the underlying algorithm just once on all data, and a naive subset method which simply averages the same subset-specific fits used in the Subsemble instead of learning a weighted combination.

We used four underlying algorithms: linear regression, lasso, regression tree, and random forest. For each of the four algorithms, we first fit the algorithm just once on the training set (the 'Full' fit).

We then divided the training set into 2, 3, 4, and 5 subsets. For each subset division, we fit each of the four algorithms on the subsets, and combined the results across the subsets in two ways: naive simple averaging across the subset-specific fits, and the Subsemble procedure.

For the simulated datasets, we simulated training sets of 1,000 observations and test sets of 10,000 observations, and repeated the experiment 10 times. For the real datasets, we split the datasets into 10 folds, and let each fold serve as the test set. Mean Squared Prediction Error (MSPE) results were averaged across the 10 trials for both simulated and real datasets. We also performed a t-test for the difference in means between each subset method (naive and Subsemble, for each number of subsets) and the 'Full' fit. Results are presented in Table 1.

With simulated dataset 1, for both linear regression and lasso, the full algorithm fit, Subsembles, and naive versions have essentially the same performance. For regression tree and random forest, all the Subsembles significantly outperform the full fit. For regression tree, the naive versions have essentially the same performance as the corresponding Subsembles, and also significantly outperform the full fit. However, for random forest, the naive versions are much worse than the Subsembles, and the naive versions perform significantly worse than the full fit.

For simulated dataset 2, the lasso once again has essentially the same performance across the full fit, Subsembles, and naive versions. With linear regression, regression tree, and random forest, the Subsembles significantly outperform the full fit. The naive version has poorer performance. With linear regression and random forest, the naive version is significantly worse than the full fit. With regression tree, the naive version does significantly improve on the full fit, but still has much worse performance than the Subsembles. In this simulation, we also see an important problem with the naive version: there is no way to account for a poor subset-specific fit. This is likely the reason why the MSPE results for the naive versions with linear regression are so high.

With the yacht dataset, the full fit of the regression tree fit was significantly better than both the Subsembles and the naive versions. For the other three underlying algorithms, at least one Subsemble significantly outperformed the full fit, while the naive versions were either not significantly different, or had significantly worse performance than the full fit.

For the diamond dataset, with linear regression and lasso, most Subsembles and naive versions has significantly better performance than the full fit, with the Subsembles being more significantly better. With regression tree, one Subsemble was significantly better than the full fit, while all naive versions were not significantly different. With random forest, both Subsembles and naive versions were significantly worse than the full fit.

Across all the datasets, we see that the Subsembles can often significantly outperform the full algorithm. Note that performance of the Subsemble depends on both the underlying algorithm and the distribution generating the data. None of the underlying algorithms always had the best performance by using the full fit, or by using Subsembles. As a result, for real datasets in which the generating distribution is unknown, we cannot predict ahead of time whether the full fit or

Table 1: *MSPE comparison results for the same underlying algorithm fit in three different ways: the 'Full' fit from fitting the algorithm only once on the entire dataset, Subsembles with two through five subsets, and a naive average of the subsets used in the Subsembles. The underlying algorithm used in each row appears in the Algorithm column. J indicates the number of subsets. The method with lowest MSPE for each underlying algorithm is in bold. The number of symbols in the superscript indicates the significance level of a t-test for the difference in means between the subset method and the full fit: 0.10 (1 symbol), 0.05 (2 symbols), 0.01 (3 symbols). Asterisks (*) are used when the subset method MSPE is significantly lower, and tick marks (′) are used when the subset method MSPE is significantly higher.*

Dataset	Algorithm	Full	Method	$J = 2$	$J = 3$	$J = 4$	$J = 5$
Sim 1							
	Linear	**347.6**	Subsemble	**347.6**	347.8	347.7	348.1
			Naive	**347.6**	348.0	347.7	348.2
	Lasso	**341.4**	Subsemble	341.6	342.0	342.2	343.1
			Naive	342.7	343.6	345.5	347.6′
	Tree	265.6	Subsemble	254.7**	253.6***	253.6***	**249.9*****
			Naive	254.9**	253.2***	254.4***	251.7***
	Forest	229.3	Subsemble	**195.1*****	195.6***	196.5***	198.5***
			Naive	246.5‴	258.4‴	270.3‴	279.6‴
Sim 2							
	Linear	340.8	Subsemble	271.7***	271.7***	**271.4*****	277.6***
			Naive	362.9‴	408.5‴	549.2‴	3.10 e6‴
	Lasso	274.0	Subsemble	274.1	**273.8**	274.2	275.1
			Naive	**273.8**	274.1	273.9	274.1
	Tree	349.9	Subsemble	271.1***	**270.9*****	**270.9*****	271.7***
			Naive	334.4***	316.8***	302.3***	295.1***
	Forest	263.0	Subsemble	**252.6*****	253.3***	253.7***	255.1***
			Naive	264.4	265.4″	266.2‴	267.0‴
Yacht							
	Linear	83.42	Subsemble	57.95*	58.18*	57.38*	**55.66****
			Naive	72.44	72.17	71.49	72.17
	Lasso	80.82	Subsemble	57.34	58.94	58.01	**55.71***
			Naive	74.17	74.74	75.15	75.16
	Tree	**4.296**	Subsemble	6.866	15.60‴	22.39‴	17.52‴
			Naive	7.349	18.75‴	24.30‴	20.41‴
	Forest	14.54	Subsemble	**7.213***	8.460	8.760	8.977
			Naive	21.13	28.28	35.35″	43.29″
Diamond							
	Linear	3.07 e5	Subsemble	**2.61 e5****	2.73 e5*	2.67 e5*	2.72 e5*
			Naive	2.74 e5*	2.75 e5*	2.94 e5	2.76 e5
	Lasso	3.13 e5	Subsemble	**2.73 e5*****	2.75 e5**	2.74 e5***	2.96 e5
			Naive	2.78 e5**	2.91 e5**	3.05 e5	2.90 e5
	Tree	1.15 e6	Subsemble	1.10 e6	**1.01 e6***	1.10 e6	1.07 e6
			Naive	1.11 e6	1.06 e6	1.18 e6	1.13 e6
	Forest	**5.05 e5**	Subsemble	6.00 e5″	6.81 e5‴	7.80 e5‴	8.26 e5‴
			Naive	6.54 e5‴	7.50 e5‴	8.04 e5‴	8.60 e5‴

Subsembles of a given underlying algorithm will have better performance.

Subsembles also perform at least as well as, and usually better than, the corresponding naive averaging versions. This result is not only practical: it is also predicted by the theoretical oracle inequality in Section 2.2. The oracle result tells us that Subsemble performs as well as the best possible combination of subset-specific fits. Since naive averaging *is* a possible combination of subset-specific fits, it follows that Subsemble is asymptotically superior.

4.3 SuperLearner Performance Comparison

In this study, we compare the performance of the SuperLearner using two different libraries of candidate algorithms: a library including only algorithms fit on the full dataset, and a library including both Subsembles and algorithms fit on the full dataset. We again used the following algorithms: linear regression, lasso, regression tree, and random forest. In the library with Subsembles versions, we included Subsembles with 2 and 5 subsets for each of the 4 algorithms, as well as the full algorithms.

For the simulated datasets, we simulated training sets of 1,000 observations and test sets of 10,000 observations, and repeated the experiment 10 times. For the real datasets, we split the datasets into 10 folds, and let each fold serve as the test set. Mean Squared Prediction Error (MSPE) results were averaged across the 10 trials for both simulated and real datasets. We also performed a t-test for the difference in means between the two SuperLearner library results. Results are presented in Table 2.

Across all datasets, the SuperLearner whose library included Subsembles outperformed the SuperLearner whose library used only full algorithm versions.

Table 2: *MSPE comparison results for SuperLearners with two different libraries: one using only algorithms fit once on the entire dataset, and the other using both algorithms fit once on the entire dataset and two Subsemble versions of each algorithm. Underlying algorithms used were: linear regression, lasso, regression tree, and random forest. The method with lowest MSPE for each each dataset is in bold. The Significance column indicates the significance level of a t-test for the difference in means between the two methods.*

Dataset	No Subsembles	Subsembles	Significance
Sim 1	228.4	**194.7**	< 0.01
Sim 2	263.9	**250.7**	< 0.01
Yacht	4.827	**4.046**	0.07
Diamond	284171	**248882**	0.02

5 Discussion

In this paper, we introduced the Subsemble procedure for fitting the same underlying algorithm on different subsets of observations, and learning the optimal weighted combination using V-fold cross-validation. We provided a theoretical statistical result, showing that Subsemble performs as well as the best possible combination of the subset-specific fits. Through simulation studies and real data analysis, we illustrated that Subsemble can provide practical performance improvements on moderate sized datasets.

Applying Subsemble to large-scale datasets is a promising direction for future research. There are a variety of details that need to be considered. For example, Subsemble's computations on each subset are independent, even in the cross-validation training steps, and thus can be easily parallelized. However, minimizing the cross-validated risk to learn the optimal combination of the subset-specific fits requires access to all the data. Single-split cross-validation, where a separate set of reserved observations is used instead of V-fold cross-validation, is one option. Selecting the number of subsets to use in a computationally friendly way is another research challenge, since using SuperLearner may not be feasible with big data. We plan to follow the work presented in this paper with future study to address these practical challenges for using Subsemble with big data.

Another topic we plan to explore in future work is the selection of subsets in Subsemble. In this paper, Subsemble's subsets were selected randomly, but other methods of obtaining the subsets are possible. For example, the data could first be clustered into J clusters, and these would also form the J subsets. Forcing the subsets to be more similar internally and/or more different from each other should result in more varied subset-specific fits. We plan to study whether doing so would improve prediction performance.

Funding

This work was supported by the National Science Foundation (Graduate Research Fellowship to Stephanie Sapp), and the National Institutes of Health (R01 AI074345-06 to Mark J. van der Laan).

References

Bache, K. and Lichman, M. (2013). UCI Machine Learning Repository.

Breiman, L. (1996a). Bagging predictors. Machine Learning *24*, 123–140.

Breiman, L. (1996b). Stacked regressions. Machine Learning *24*, 49–64.

Chu, S. (2001). Pricing the Cs of diamond stones. Journal of Statistical Education *9*.

Lin, J. and Kolcz, A. (2012). Large-scale machine learning at twitter. In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data SIGMOD '12 pp. 793–804, ACM, New York, NY, USA.

van der Laan, M. J., Dudoit, S. and van der Vaart, A. W. (2006). The cross-validated adaptive epsilon-net estimator. Statistics and Decisions *24*, 373–395.

van der Laan, M. J., Polley, E. C. and Hubbard, A. E. (2007). Super Learner. Statistical Applications in Genetics and Molecular Biology *6*.

Wolpert, D. H. (1992). Stacked generalization. Neural Networks *5*, 241–259.

Zhang, Y., Duchi, J. C. and Wainwright, M. (2012). Comunication-Efficient Algorithms for Statistical Optimization. Technical report arXiv:1209.4129 [stat.ML].