University of California, Berkeley

U.C. Berkeley Division of Biostatistics Working Paper Series

Year 2014 *Paper* 321

A Scalable Supervised Subsemble Prediction Algorithm

Stephanie Sapp* Mark J. van der Laan†

*University of California, Berkeley, Department of Statistics, sapp@berkeley.edu

†University of California, Berkeley, School of Public Health, Division of Biostatistics, laan@berkeley.edu

A Scalable Supervised Subsemble Prediction Algorithm

Stephanie Sapp and Mark J. van der Laan

Abstract

<blockquote>Subsemble is a flexible ensemble method that partitions a full data set into subsets of observations, fits the same algorithm on each subset, and uses a tailored form of V-fold cross-validation to construct a prediction function that combines the subset-specific fits with a second metalearner algorithm. Previous work studied the performance of Subsemble with subsets created randomly, and showed that these types of Subsembles often result in better prediction performance than the underlying algorithm fit just once on the full dataset. Since the final Subsemble estimator varies depending on the data used to create the subset-specific fits, different strategies for creating the subsets used in Subsemble result in different Subsembles. We propose supervised partitioning of the covariate space to create the subsets used in Subsemble, and using a form of histogram regression as the metalearner used to combine the subset-specific fits. We discuss applications to large-scale data sets, and develop a practical Supervised Subsemble method using regression trees to both create the covariate space partitioning, and select the number of subsets used in Subsemble. Through simulations and real data analysis, we show that this subset creation method can have better prediction performance than the random subset version.</blockquote>

1 Introduction

Procedures using subsets of observations from a full available data set are promising tools for prediction with massive data sets. By operating on subsets of observations, computations can take advantage of modern parallel computational resources. Much recent research work has focused on proposing and evaluating the performance of different subset prediction procedures.

A classic subset prediction method is bagging, or bootstrap aggregating, developed in Breiman (1996). In bagging, one subsamples a large number of fixed size bootstrap samples, fits the same prediction algorithm on each bootstrap sample. The final prediction function is given by the simple average of the subset-specific fits.

Recently, Zhang et al. (2013) proposed two subset methods for estimating the parameter of a parametric prediction model: an average mixture (AVGM) procedure, and a bootstrap average mixture (BAVGM) procedure. Both procedures first partition the full data set into disjoint subsets, and estimate the parameter of interest within each subset. To obtain the final parameter estimate, AVGM takes a simple average of the subset-specific estimates. BAVGM first draws 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. To obtain the final parameter estimate, BAVGM takes a simple average of the subset-specific bootstrap bias-corrected estimates.

Another recent classification method using subsets was discussed in Lin and Kolcz (2012). This case study explored using subsets of observations to train classification algorithms, and combining the results linearly. The authors mention the possibility of weighting each classifier if different underlying algorithms are used, but recommend simple averaging if the same underlying classifier is trained on different subsets of observations.

A key drawback of each of the above subset methods is that they do not differentiate between the quality of each subset-specific fit. To address this, Sapp et al. (2014) proposed the Subsemble method. Subsemble is a general subset ensemble prediction algorithm that partitions a full data set into subsets of observations, fits one or more underlying algorithm on each subset, and combines the subset-specific fits through a second metalearner algorithm using a clever form of V-fold cross-validation, thus accounting for the quality of each subset-specific fit. Previous work studied the performance of Subsembles created from randomly selected subsets of observations. In particular, Sapp et al. (2014) demonstrated that such random subset Subsembles have good performance in practice, and often provide better prediction performance than fitting the underlying algorithm only once on a full available data set.

Note that the final Subsemble estimator varies depending on the data used to create each subset-specific fit. As a result, different strategies for creating Subsemble's subsets will result in different Subsembles. In this paper, we introduce a different method for partitioning a data set into the subsets used in Subsemble. In particular, we propose the use of Supervised Subsembles, which create subsets through supervised partitioning of the covariate space, combined with a form of

histogram regression as the metalearner used to combine these subset-specific fits. We also develop a practical Supervised Subsemble method, which employs regression trees to both partition the observations into the subsets used in Subsemble, and select the number of subsets to use. We discuss computational independence properties of our proposed methods that are advantageous for applications involving big data, and show through simulations that our proposed version of Subsemble can result in further improved prediction performance.

The remainder of our paper is organized as follows. We review the Subsemble method in Section 2. The Supervised Subsemble approach for creating subsets, along with associated metalearner, is presented in Section 3. We describe the practical Supervised Subsemble method using regression trees in Section 4. Simulation and real data analysis results are discussed in Section 5. We summarize and conclude in Section 6.

2 The Subsemble method

Subsemble is a general subset ensemble prediction algorithm that partitions a full data set into subsets of observations, fits the same underlying algorithm on each subset, and combines the subset-specific fits through a second metalearner algorithm using a clever form of V-fold cross-validation.

Subsemble begins by partitioning n iid training observations (X_i, Y_i) into J disjoint subsets. Next, Subsemble fits the same underlying prediction algorithm $\hat{\Psi}$ on each subset, resulting in J subset-specific estimators $\hat{\Psi}_j$, where $\hat{\Psi}_j$ is defined as $\hat{\Psi}$ applied to the j-th subset. Finally, Subsemble combines the subset-specific fits by minimizing cross-validated risk with a second metalearner algorithm $\hat{\Phi}$, through a tailored version of V-fold cross-validation.

The V folds used in Subsemble are constructed to preserve the assignment of observations to subsets used in the J subset-specific estimators. Instead of simply partitioning the n observations into V folds, we instead partition each individual subset into V folds. The optimal combination of the subset-specific fits is then determined by applying the metalearner algorithm $\hat{\Phi}$ to the n redefined observations $(\tilde{X}_i, Y_i)$, where the redefined input vector $\tilde{X}_i = \{\hat{\Psi}_{j,v(i)}\}_{j=1}^{J}(X_i)$ consists of the J predicted values obtained by evaluating, at X_i, the J subset-specific estimators trained when the fold containing observation i was left out. The final prediction function is then the best metalearner combination of the J subset-specific fits.

The Subsemble procedure has a variety of desirable properties. Any underlying algorithm and metalearner algorithm, including both parametric and nonparametric methods, can be accommodated. The quality of each subset-specific fit is taken into account in the final combination function. In particular, rather than taking a naive average of the subset-specific fits, Subsemble uses cross-validation to differentiate fit quality and learn the optimal weighted combination. By using cross-validation, Subsemble is also able to take advantage of the cross-validation's theoretical performance guarantees. Specifically an oracle result for Subsemble, given in Sapp et al. (2014),

provides a theoretical guarantee that Subsemble performs as well as the best possible combination of the subset-specific fits.

3 Supervised Subsemble

3.1 Supervised partitioning of the covariate space

To obtain the subsets used in Subsemble, we propose partitioning of the covariate space to create J disjoint subsets of covariates, $S_1, \ldots, S_J$, such that any given vector of covariates belongs to exactly one of the J subsets. Numerous methods to achieve a partitioning of the covariate space are available. For example, the unsupervised k-means algorithm could be used to first cluster the observations into J clusters based on only their covariates, with these J clusters also forming the J subsets. In general, supervised partitioning methods also consider the outcome variable, thus creating a partitioning that is directly predictive of the outcome.

Compared to randomly selecting the subsets, constructing the subsets to be similar internally and different from each other results in locally smoothed subset-specific fits when fitting the same algorithm on each subset. In particular, each subset-specific fit is in fact tailored for the associated partition of the covariate space. More specifically, the subset-specific fit $\hat{\Psi}_j$ associated with S_j is tailored for the partition S_j, and we would not expect $\hat{\Psi}_j$ to be a good fit for observations with covariates in some other $S_{j'}$, where $j \neq j'$.

3.2 Modified histogram regression metalearner

To reflect the above observation, we propose using a modified version of histogram regression as the metalearner used to combine the subset-specific fits. The usual form of histogram regression, applied to our J subsets S_j, would output a local average of the outcome Y within each subset. Instead of this standard form of histogram regression, our version of histogram regression outputs the associated $\hat{\Psi}_j$ for each subset S_j. In addition, our version of histogram regression includes a coefficient and intercept within each subset. Concretely, we define our proposed histogram regression metalearner $\hat{\Phi}$ for combining the subset-specific fits as follows:

$$\hat{\Phi}(\hat{\Psi})(x) = \sum_{j=1}^{J} I(x \in S_j)\left(\beta_j^0 + \beta_j^1 \hat{\Psi}_j(x)\right) \tag{1}$$

The value of the functional form of Equation 1 is its generalization to using more than one underlying algorithm. That is, applying multiple prediction algorithms to each subset. That is, instead

of applying a single underlying algorithm $\hat{\Psi}$ to each subset, the Subsemble procedure readily accommodates applying L underlying algorithms $\hat{\Psi}^1, \ldots, \hat{\Psi}^L$ to each subset. The generalization of Equation 1 gives us the following histogram regression metalearner $\hat{\Phi}$ for combining these multiple subset-specific fits as follows:

$$\hat{\Phi}(\hat{\Psi}^1, \ldots, \hat{\Psi}^L)(x) = \sum_{j=1}^{J} \left[I(x \in S_j) \left(\beta_j^0 + \sum_{\ell=1}^{L} \beta_j^\ell \hat{\Psi}_j^\ell(x) \right) \right] \tag{2}$$

3.3 Computational independence of subsets

Note that the computations Subsemble performs on each subset are *always* independent, even in the cross-validation training steps. This is because the partitioning of the n observations into J subsets remains the same during cross-validation. As a result, leaving out a fold v from a subset j produces all the data assigned to the j-th subset in the cross-validation training set. However, with randomly constructed subsets, minimizing the cross-validated risk to learn the optimal combination of the subset-specific fits requires access to all the data.

The Supervised Subsembles proposed here have the additional benefit of preserving the subset computation independence. That is, if subsets are known a priori, by keeping the subset assignments fixed, computations on the subsets of the Supervised Subsemble described in this section remain completely computationally independent across the entire procedure.

To see this, let β_n to be the cross-validation selector of β. Then by definition of the cross-validation selector,

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

Using the fact that each observation i belongs to exactly one subset S_j, we rearrange terms as follows:

$$= \arg\min_{\beta} \sum_{j=1}^{J} \sum_{i:i \in S_j} \left\{ Y_i - \hat{\Phi}_\beta(\tilde{X}_i) \right\}^2$$

From the definitions of $\hat{\Phi}$ and $\tilde{X}_i$,

$$= \arg\min_{\beta} \sum_{j=1}^{J} \sum_{i:i \in S_j} \left\{ Y_i - \sum_{j'=1}^{J} \left[I(X_i \in S_{j'}) \left(\beta_{j'}^0 + \sum_{\ell=1}^{L} \beta_{j'}^\ell \hat{\Psi}_{j',v(i)}^\ell(X_i) \right) \right] \right\}^2$$

Again, since each observation i belongs to exactly one S_j,

$$= \arg\min_{\beta} \sum_{j=1}^{J} \sum_{i:i \in S_j} \left\{ Y_i - \left[\beta_j^0 + \sum_{\ell=1}^{L} \beta_j^\ell \hat{\Psi}_{j,v(i)}^\ell(X_i) \right] \right\}^2$$

Finally, since the observations $i : i \in S_j$ are disjoint for different j, terms involving each β_j can be minimized independently:

$$= \left\{ \arg\min_{\beta_j} \sum_{i:i\in S_j} \left(Y_i - \left[\beta_j^0 + \sum_{\ell=1}^{L} \beta_j^\ell \hat{\Psi}^\ell_{j,v(i)}(X_i) \right] \right)^2 \right\}_{j=1}^{J}$$

Thus, each term β_j can be estimated by minimizing cross-validated risk using only the data in subset S_j.

We are thus able to estimate the coefficient associated with each subset independently, using only data within that subset. Consequently, unlike the randomly constructed subsets, we avoid needing to recombine data to produce the final prediction function.

4 SRT Subsemble: Supervised Regression Tree Subsemble

4.1 Motivation

The Supervised Subsembles proposed in the previous section maintain computational independence across subsets. However, this assumes that both the number of subsets, and the partitioning of the covariate space, are provided. In this section, we propose the a practical Supervised Regression Tree Subsemble (SRT Subsemble) algorithm, which uses regression trees with Subsemble as a practical and computationally feasible way to determine both of the number of subsets, and the partitioning of the covariate space to create the subsets.

To motivate our approach, we first discuss relevant concerns about constructing covariate space partitions when dealing with large-scale data set applications. For big data, splitting data up can be a significant computational concern. As a result, it is preferable to avoid approaches that, when creating different numbers of subsets, first split the full data set into, for example, two partitions, and then recombine the data in order to determine a way to split the data into three partitions. Instead, it is better to work with greedy methods, which enforce that once the data has been split, any further splits will only divide an already existing partition.

4.2 Constructing and selecting the number of subsets

Classification and Regression Trees (CART), developed by Breiman et al. (1984), recursively partition the covariate space by creating binary splits of one covariate at a time. Concretely, using covariate vector $X_i = (X_i^1, \ldots, X_i^K)$, the first iteration of CART selects a covariate X^k, and then creates the best partition of the data based on that covariate. As a metric to measure and select the

best covariate split for a continuous outcome, CART fits an ANOVA model and uses the between-groups sum-of-squares metric. The best split is then selected to maximize this between-groups sum-of-squares. For additional details, we refer the reader to Breiman et al. (1984).

The first iteration of CART thus creates the first partition of the data based on two regions $S_1^1 = I(X^k \leq c_1)$ and $S_1^2 = I(X^k > c_1)$. Subsequent splits are obtained greedily, by repeating this procedure on each new partition. For example, the second iteration of CART selects a covariate $X^{k'}$, and partitions S_1^1 into $S_2^1 = I(X^k \leq c_1,\ X^{k'} \leq c_2)$ and $S_2^2 = I(X^k \leq c_1,\ X^{k'} > c_2)$. For a given partitioning, the standard prediction function from CART outputs the local average of the outcome Y within each subset.

To partition the covariate space and select the number of subsets for Subsemble, we apply CART as follows. First, we simply run the CART algorithm on the data set, resulting in a sequence of nested partitionings of the covariate space. That is, the CART algorithm outputs a sequence of sub-trees: a first tree with a single root node, a second tree with two nodes, a third tree with three nodes, and so on, ending with the full tree with M nodes. We treat the m nodes of each m-th sub-tree as a candidate partitioning into m subsets.

Next, we explore the sequence of M possible partitions (sequence of sub-trees) produced by CART, beginning at the root. For each candidate number of subsets $1,\ldots,M$, we fit the associated Supervised Subsemble. Concretely, with m subsets, we create L subset-specific fits $\hat{\Psi}_j^\ell$ for each subset $j = 1,\ldots,m$, and create the overall prediction function according to Equation 2. Note that we use CART to create the subsets S_j that appear in Equation 2. Finally, we select the best number of subsets as the Subsemble with minimum cross-validated risk.

4.3 Computational advantages

Our proposed SRT Subsemble retains the desirable subset computational independence discussed in Section 3, while also creating the subsets used in Subsemble in a computationally friendly way, as well as providing a criteria for choosing the number of subsets to use in Subsemble.

As a consequence of this subset computational independence, note that in fitting a sequence of Subsembles in a series of sub-trees, each subsequent Subsemble only requires computation for the two new nodes at each step. That is, given the Subsemble fit with, say, m subsets, computing the next Subsemble with $m+1$ subsets only requires computation for the two new nodes formed in the $m+1$-st split of the tree. This is due the fact that the nodes are computationally independent in the SRT Subsemble framework, plus the fact that at each split of the tree, all nodes remain the same, except for the single node in the m-th tree that is split into two new nodes in the $m+1$-st tree.

4.4 Implementation flexibility

There are several paths that can be taken when applying the SRT Subsemble process in practice. These user-selected options allow SRT Subsemble to be quite flexible, with decisions made to suit the application at hand. There is no one best approach, instead the options will be determined based on the application/constraints/desired properties. We briefly discuss these options.

First, the user must decide how to build and explore the tree. One possibility is to simply build a very large tree, resulting in a full tree with M nodes, build a Subsemble for each sub-tree $1,\ldots,M$, and through this process simply locate the Subsemble with the lowest cross-validated risk among the sequence of sub-trees outputted by CART. Alternatively, a greedy process can be used. Instead of calculating cross-validated risk for all sub-trees of a very large tree, the cross-validated risk can be tracked while the tree is being built-out. That is, after each additional split to the tree, build the associated Subsemble, calculate its associated cross-validated risk, and stop adding additional splits when some stopping criteria is achieved. As an example, the stopping criteria could be an increase of the cross-validated risk.

Second, the user must decide where in the tree to start building Subsembles. The most obvious approach is to start with building a Subsemble at the root node of the tree. That is, building a Subsemble with only one subset containing all observations. For small- to moderate-sized data sets, where computational considerations are of less concern, this is a good choice. However, for large-scale data sets, it may be preferable to first split the data into partitions of some desired size, and only then start building Subsembles. This approach would allow the user to take advantage of multiple independent computational resources, since each partition of data could be transferred to a dedicated computational resource, since all subsequent computations remain independent from other partitions.

5 Data analysis

5.1 Preliminary observations

We next present results from comparing the performance of SRT Subsemble with the version of Subsemble using randomly created subsets. We discuss further implementation details in the next subsection.

Before presenting our simulated and real data set results, we first discuss a few preliminary observations, through simple examples, which show that there are certainly scenarios in which SRT Subsemble results in better performance than the random subset version of Subsemble. To see this, consider an actual histogram with a single covariate, where the outcome is simply the mean in various subsets of that covariate. Further, suppose we use a single underlying algorithm $\hat{\Psi}$,

which simply takes the mean outcome among observations. In this case, it is clear that the SRT Subsemble procedure will produce superior prediction performance, as compared to the random subset version of Subsemble.

We also note that using homogeneous subsets presents desirable behavior for more subsets with fewer observations. With SRT Subsemble, as subsets become smaller, they also become more homogeneous. As a result, data-adaptive prediction algorithms can still result in good fits. In contrast, when using random subsets, aggressive data-adaptive algorithms often sacrifice performance from over-fitting when subsets become too small. We also note that using homogeneous subsets allow less data-adaptive prediction algorithms to achieve good fits within each subset. In particular, we do not need to use aggressive algorithms within each subset to get a good fit.

5.2 Implementation details

As mentioned above, we compared the performance of SRT Subsemble with the version of Subsemble using randomly created subsets. In both methods, we estimated β coefficients to minimize cross-validated risk, and used the same four underlying algorithms for the purposes of this demonstration: linear regression, lasso, regression tree, and random forest. Additional details about the methods compared are discussed below.

For SRT Subsemble, we used the following procedure to build the tree used in SRT Subsemble. We used the R package `rpart` (Therneau et al. (2012)) to build a full tree. In the `rpart` package, the splits and size of the tree built are controlled by four parameters: `minsplit` (minimum number of observations needed for a node to split), `minbucket` (minimum number of observations in any node), `cp` (complexity parameter), and `maxdepth` (maximum depth of any node in the tree). Since we used $V = 10$-fold cross-validation to select the optimal number of subsets, we set `minbucket`$=2V$ to ensure a sufficient number of observations in each node to perform this cross-validation. For the remaining parameters, we used `rpart`'s defaults: `minsplit` $= 3\times$`minbucket`, `cp` $= 0.01$, and `maxdepth` $= 30$.

We then selected the best number of subsets for the SRT Subsemble by building a Subsemble as in Equation 2 at each sub-tree outputted by `rpart`, starting at the root node with only one subset, and selecting the Subsemble with the lowest estimated cross-validated risk among the sequence of sub-trees outputted by `rpart`.

For the random subset Subsembles, we first built the same number of Subsembles as those that we explored with the SRT Subsemble. That is, if the SRT Subsemble explored a full tree with M nodes, we built random subset Subsembles with $1,\ldots,M$ nodes. We then combined the subset-specific fits according the the following equation:

$$\hat{\Phi}(\hat{\Psi}^1,\ldots,\hat{\Psi}^L)(x) = \sum_{j=1}^{J}\left(\beta_j^0 + \sum_{\ell=1}^{L}\beta_j^{\ell}\hat{\Psi}_j^{\ell}(x)\right)$$

To select the optimal number of subsets for the finally selected random subset version, we simply selected the Subsemble with the lowest *oracle* risk; that is, the lowest true risk on the test data. Note that this is not possible in practice, and is included here only for illustration purposes. Observe that we are comparing the SRT Subsemble to the *best conceivable* random subset version of Subsemble. In particular, this includes the version with only a single subset: a combination of the underlying algorithms fit on the full available data set (i.e., the SuperLearner method of van der Laan et al. (2007)).

5.3 Description of data sets

In the studies that follow, we used four small- to moderate-sized data sets (Synthetic 1, Synthetic 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. We created this pair of synthetic data to demonstrate one scenario in which the SRT Subsemble provides better performance (Synth 1), and another scenario in which the random subset Subsemble yields better performance (Synth 2). The first synthetic dataset exhibits significant greater local behavior than the second synthetic dataset. Both simulated data sets have 2 input variables $X_1, X_2 \sim N(0,9)$, and random error $\varepsilon \sim N(0,1)$. The outcome variable for each simulated data set is generated as follows:

Synth 1:

$$Y = \varepsilon + \sin(X_1) + 2\log(|X_1|) + 3\sqrt{|X_1|} + \sin(0.5\pi X_1)$$

Synth 2:

$$Y = 2 + \varepsilon + \sin(X_1)$$

Note that in practical applications, we do not know the true data generating distribution a priori. In particular, for datasets with many covariates, it becomes challenging even to visualize the data, and thus impractical to determine ahead of time whether or not the data exhibits significant local behavior. As a result, the two synthetic data sets presented here merely serve for illustrative purposes, to provide the reader with some general intuition for the performance of the SRT Subsemble method.

The second two datasets are publicly available real-world data, and are the same data sets studied in the analysis of Sapp et al. (2014). These data sets illustrate actual data from applications in which the SRT Subsemble method performs better than the random subset Subsemble. 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.

5.4 Results

For the synthetic data sets, we simulated training and test sets of 1,000 observations, and repeated the experiment 10 times. For the real data sets, we split the data sets 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 methods. The average number of subsets used in each method, as well as the maximum number of subsets (size of the full tree built) are also included. Results are presented in Table 1.

Table 1: *MSPE comparison results for SRT Subsemble, and oracle-selected version of Subsemble with random subsets. 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 Sig / Max column indicates the significance level of a t-test for the difference in means between the two methods, and the maximum number of subsets considered.*

Dataset		SRT	Random Oracle	Sig / Max
Synth 1	MSPE	**1.21**	1.45	$<$0.01
	# Subsets	6.3	2.5	7.5
Synth 2	MSPE	2.84	**1.40**	$<$0.01
	# Subsets	7.4	1.4	9.8
Yacht	MSPE	**1.19**	2.96	0.01
	# Subsets	3.2	1.7	3.5
Diamond	MSPE	**1.30 e5**	2.10 e5	$<$0.01
	# Subsets	3.1	2.1	5.0

From Table 1, we see that the SRT Subsemble method significantly performs better than the oracle-selected random subset version on three of the four data sets: synthetic dataset 1, the yacht data, and the diamond data. In fact, for all datasets, the two methods are significantly different at the 0.01 level.

We emphasize that while the SRT Subsemble method is viable in practice, the comparison oracle-selected random subset Subsemble is not, since it utilizes the test data to select the number of subsets to minimize the MSPE *on the test data*. The fact that the SRT Subsemble method achieves lower MPSE than this oracle-selected random subset version show clearly that SRT Subsemble can often significantly outperform any random subset version, including a single subset.

6 Discussion

In this paper, we introduced Supervised Subsembles and proposed the SRT Subsemble method. Supervised Subsembles partition the covariate space to obtain subsets, use a modified form of

histogram regression as the metalearner used to combine the subset-specific fits. We described how Supervised Subsembles preserve computational independence of the subsets throughout the entire algorithm. SRT Subsemble is a practical method to both construct the covariate partitioning, and select the optimal number of subsets. We explained the desirable computational properties of SRT Subsemble, as well as the flexibility it provides in implementation. Finally, we presented simulated and real data set results, which demonstrated that the SRT Subsemble method can result in better prediction performance than comparable random subset Subsembles.

Future research should explore more practical suggestions and concrete recommendations for implementing and using SRT Subsemble in practice on large-scale datasets. For example, with big data, it is probably impractical to build a very large tree. While we give some ideas in this paper regarding stopping criteria, additional work should be done to determine improved stopping criteria. Determining the starting node is another interesting area for future work. That is, for very large data sets, starting at the root node is likely not feasible. As a result, future study should examine criteria for selecting this starting point.

Modifying the CART algorithm used in SRT Subsemble is another area for future work. Rather than simply using the default CART algorithm to build a tree, and then exploring the Subsembles associated with the resulting sub-trees, future work should consider instead directly incorporating Subsembles into the CART procedure. For example, future work could consider using a different metric to determine the best covariate split, such as the associated Subsemble's cross-validated risk.

Funding

This work was supported by the National Science Foundation (Graduate Research Fellowship to Stephanie Sapp), and the National Institutes of Health (grant R01 AI074345-06A1 to Mark J. van der Laan).

References

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

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

Breiman, L., Friedman, J., Olshen, R. and Stone, C. (1984). Classification and Regression Trees. Wadsworth and Brooks, Monterey, CA.

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.

Sapp, S., van der Laan, M. J. and Canny, J. (2014). Subsemble: an ensemble method for combining subset-specific algorithm fits. Journal of Applied Statistics *41*, 1247–1259.

Therneau, T., Atkinson, B. and Ripley, B. (2012). rpart: Recursive Partitioning. R package version 4.1-0.

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

Zhang, Y., Duchi, J. C. and Wainwright, M. J. (2013). Comunication-Efficient Algorithms for Statistical Optimization. Journal of Machine Learning Research *14*, 3321–3363.