# Supervised neighborhoods for distributed nonparametric regression

**Adam Bloniarz**
Dept. of Statistics
UC Berkeley

**Christopher Wu**
Computer Science Dept.
UCLA

**Bin Yu**
Dept. of Statistics
Dept. of EECS
UC Berkeley

**Ameet Talwalkar**
Computer Science Dept.
UCLA

## Abstract

Techniques for nonparametric regression based on fitting small-scale local models at prediction time have long been studied in statistics and pattern recognition, but have received less attention in modern large-scale machine learning applications. In practice, such methods are generally applied to low-dimensional problems, but may falter with high-dimensional predictors if they use a Euclidean distance-based kernel. We propose a new method, SILO, for fitting prediction-time local models that uses supervised neighborhoods that adapt to the local shape of the regression surface. To learn such neighborhoods, we use a weight function between points derived from random forests. We prove the consistency of SILO, and demonstrate through simulations and real data that our method works well in both the serial and distributed settings. In the latter case, SILO learns the weighting function in a divide-and-conquer manner, entirely avoiding communication at training time.

## 1 INTRODUCTION

Modern datasets collected via the internet and in scientific domains hold the promise for a variety of transformational applications. Non-parametric methods present an attractive modeling choice for learning from these massive and complex datasets. While parametric methods that optimize over a fixed function space can suffer from high approximation error when the task is complex, non-parametric methods augment the capacity of the underlying statistical model as the datasets grows in scale and diversity. Moreover, the scale of these datasets makes non-parametric estimation feasible even when considering minimax rates for non-parametric functions that require the data to scale exponentially in the feature dimension.

Global non-parametric models, e.g., neural networks and kernel methods, learn complex response surfaces and can be quite expensive to train on large-scale datasets. Both machine learning and computer systems researchers are actively exploring techniques to efficiently embed these learning approaches into parallel and distributed computing frameworks, e.g., [1]–[5]. In contrast, local models are simple, interpretable, and provide an attractive computational profile, by potentially drastically reducing training time at the expense of a moderate increase in test time.

However, to date local methods have been rarely employed on large-scale learning tasks due to both statistical and computational concerns. Statistically, classical methods struggle with even a moderate number of features due to the curse of dimensionality. Although these local methods are minimax optimal, this minimax rate is quite conservative when the response does not involve all dimensions. Although local approaches relying on 'supervised neighborhoods,' e.g., DANN [6], CART [7], Random Forests [8], and Rodeo [9], demonstrate empirical and theoretical success at overcoming this dimensionality issue, they do so at great computational expense.

In this work, we address these statistical and computional issues, focusing on the problem of non-parametric regression. We propose a novel family of **S**upervi**s**ed **Lo**cal modeling methods (SILO) that build on the interpretation of random forests as a local model to identify supervised neighbors. Like $k$-NN methods, the width of our neighborhoods adapts to the local density of data: higher density leads to smaller neighborhoods while lower density means neighborhoods spread out and borrow strength from distant points. Unlike $k$-NN or local polynomial methods, our neighborhoods are determined by the shape of the re-

Appearing in Proceedings of the $19^{th}$ International Conference on Artificial Intelligence and Statistics (AISTATS) 2016, Cadiz, Spain. JMLR: W&CP volume 51. 

sponse rather than some fixed metric (hence they are supervised neighborhoods).

Additionally, we observe that datasets of the size we need to fit nonparametric functions are often stored and processed in a distributed fashion. We thus devise an efficient distributed variant of SILO, which naturally targets massive, distributed datasets. In this setting, worker nodes at training time independently calculate supervised neighborhoods using information from distinct random forests. Given a test point, the master node gathers supervised neighborhood data from all workers, and then trains a local linear model using this neighborhood data.

To summarize, our contributions are as follows:

- We introduce a novel adaptive local learning approach, SILO, based on the idea of supervised neighbors.
- We present a distributed variant of SILO that is well suited for large-scale distributed datasets.
- We prove consistency of SILO under a simple model of random forests.
- We show experimentally that SILO outperforms both standard random forests and Euclidean-distance based local methods in single node settings.
- We implement the distributed variant of SILO in Apache Spark [10], and demonstrate its favorable performance relative to the default Spark/MLlib [11] Random Forest implementation.

## 2 METHODS BASED ON LOCAL MODELS

There are three ingredients to a method for local model-based nonparametric regression: a loss function $L : \mathbb{R} \to \mathbb{R}$, a weight function $w(\cdot, \cdot)$, and a function space $\mathcal{F}$. In this work, we assume that the loss function is the squared loss, i.e., $L(u) = u^2$. The weight function is a mapping $w(\cdot, \cdot) : \mathbb{R}^p \times \mathbb{R}^p \to [0, \infty)$, which may be fixed, or may depend on the training dataset. $\mathcal{F}$ is a family of functions where $f \in \mathcal{F}$ maps $\mathbb{R}^p \to \mathbb{R}$. Generally $\mathcal{F}$ is a fairly simple function space, as it need only provide a good approximation to the response surface locally. Common function spaces are constant functions, linear functions, or higher-order polynomials of fixed degree.

We now describe the general form for local modeling methods. Assume we have a set of training data pairs, $\{(y_i, \mathbf{x}_i)\}_{i=1}^n$. Given a test point $\mathbf{x} \in \mathbb{R}^p$, we define a function $\hat{g}(\mathbf{x})$ which approximates $E(y \mid \mathbf{x})$, via the following two-step process:

$$\text{let } \hat{f}_{\mathbf{x}}(\cdot) = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{n} w(\mathbf{x}_i, \mathbf{x}) \left(y_i - f\left(\mathbf{x}_i - \mathbf{x}\right)\right)^2 \quad (1)$$

$$\hat{g}(\mathbf{x}) = \hat{f}_{\mathbf{x}}(\mathbf{0}) \,. \quad (2)$$

For instance, we can consider $k$-NN regression under this framework by first defining $\mathcal{K}_{\mathbf{x}}$ as the set of $\mathbf{x}$'s $k$ nearest neighbors. Then, $w(\mathbf{x}_i, \mathbf{x}) = 1$ if $\mathbf{x}_i \in \mathcal{K}_x$ and zero otherwise, and $\hat{f}_{\mathbf{x}}(\cdot)$ is a constant function that returns $\frac{1}{k} \sum_{x_i \in \mathcal{K}_{\mathbf{x}}} y_i$. Partitioning methods like CART [7], and random forests [8] can also be seen as examples of local models with constant $\hat{f}_{\mathbf{x}}(\cdot)$ functions.

LOESS [12] uses a fixed weighting function, and fits local polynomial functions. We describe the particular setting of linear functions in more detail in Section 3 (see equations 8 and 9). Other methods exist that are similar in spirit to the local modeling framework defined in equations 1 and 2 but do not fit precisely with our formulation, e.g., tree-based piecewise polynomials models [13], and multivariate adaptive regression splines [14].

### 2.1 Euclidean distance-based local methods

A common form for the weight function $w$ is the following:

$$w(\mathbf{x}_1, \mathbf{x}_2) = k(\|\mathbf{x}_1 - \mathbf{x}_2\|^2) \quad (3)$$

where $\|\cdot\|$ is the Euclidean, or $L^2$ norm; $k : [0, \infty) \to [0, \infty)$ is a univariate kernel function. Common examples include the rectangular, triangular, Epanechnikov, cosine, and Gaussian kernels. If the local model family $\mathcal{F}$ is chosen to be polynomials of degree $\ell$, such an estimator is denoted as an $LP(\ell)$ method (we adopt the notation of Tsybakov [15]). If the true conditional expectation function $E(y \mid \mathbf{x})$ belongs to the Hölder class $\Sigma(\beta, L)$, then, with appropriate scaling of the kernel function, the mean square error of the $LP(\lfloor \beta \rfloor)$ estimator converges to zero at the minimax optimal rate of $n^{-\frac{2\beta}{2\beta+p}}$. In practice, despite their optimality properties, $LP(\ell)$ estimators are not generally applied to higher dimensional problems where some dimensions may be irrelevant. As an example, Figure 1 shows the empirical performance of $k$-nearest neighbors, LOESS with local linear models, and random forests on the popular Friedman1 simulation from [14]. In this simulation, $p = 10$, $\mathbf{x}$ is distributed uniformly on the hypercube in 10 dimensions, and $y = 10 \sin(\pi x_1 x_2) + 20 \left(x_3 - \frac{1}{2}\right)^2 + 10 x_4 + 5 x_5 + \epsilon$, where $\epsilon \sim \mathcal{N}(0, 9)$. Note that only 5 of the 10 dimensions of $\mathbf{x}$ are relevant to $y$. We see that random forests outperform LOESS and nearest neighbor methods; in this case the tuning parameters of LOESS and $k$-NN have been optimized on the holdout set, while

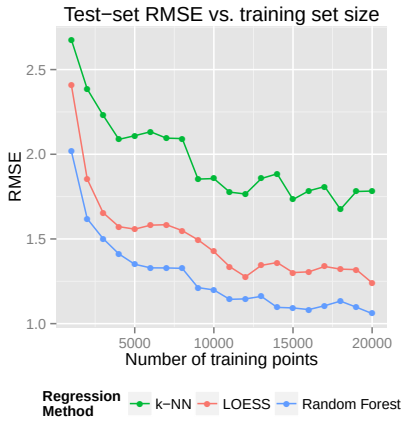


Figure 1: Comparison of non-parametric techniques on the Friedman1 simulation. Parameters of LOESS and k-NN were tuned on the holdout set, while random forest parameters were set at default values (3 candidate variables at each node, trees trained to 5 points per leaf.

the tuning parameters of random forests have been set to their default. Unlike random forests, the Euclidean distance based methods to not adapt to the fact that irrelevant predictors are included in the model, and the local neighborhoods for each test point do not 'spread out' in the flat directions. We explore this property of random forests in the next section.

### 2.2 Random forests as adaptive nearest neighbors

Soon after the introduction of the random forest algorithm, Lin and Jeon [16] observed that random forests can be considered an adaptive potential nearest neighbor method (PNN). To explain this idea, we introduce some notation. Given a random forest consisting of $K$ trees, we define $\theta$ to be the random parameter vector that determines the growth of a tree (for example, $\theta$ specifies the sampling of training points and the selection of covariates at each node). The tree built with $\theta$ is denoted as $T(\theta)$, and for $\mathbf{x} \in \mathbb{R}^p$, let $R(\mathbf{x}, \theta)$ be the rectangle corresponding to the terminal node of $T(\theta)$ containing $\mathbf{x}$. We define the connection function of a tree, which is the indicator that two points share the same terminal node of a particular tree.

$$w(\mathbf{x}_1, \mathbf{x}_2, \theta) = \mathbb{1}\left\{\mathbf{x}_1 \in R(\mathbf{x}_2, \theta)\right\} \tag{4}$$

Define the number of training points contained in a leaf-node containing the point $\mathbf{x}$, for a tree trained with parameter $\theta$ as $k_\theta(\mathbf{x}) = \sum_{i=1}^n w(\mathbf{x}_i, \mathbf{x}, \theta)$. Then

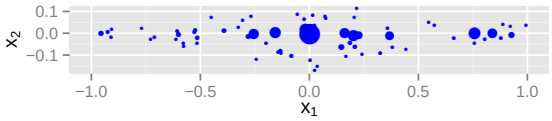


Figure 2: Weights of random forest. Each circle represents a point in the training set, and diameter of the circle represents the weight of that point when making a prediction at the origin. The function being fit does not vary along the $x_1$ coordinate.

the prediction of a random forest can be written as

$$\hat{g}(\mathbf{x}) = \frac{1}{K}\sum_{j=1}^{K}\left[\frac{\sum_{i=1}^n w(\mathbf{x}_i, \mathbf{x}, \theta_j) y_i}{k_{\theta_j}(\mathbf{x})}\right] \tag{5}$$

Note that this takes the form of equations 1 and 2, where $\mathcal{F}$ is the family of constant functions, and we define the random forest weighting function

$$w_{\mathrm{RF}}(\mathbf{x}_i, \mathbf{x}) = \frac{1}{K}\sum_{j=1}^{K}\left[\frac{w(\mathbf{x}_i, \mathbf{x}, \theta_j)}{k_{\theta_j}(\mathbf{x})}\right] \tag{6}$$

Note that this weighting function is not derived from any explicit distance metric. However, as noted in Lin and Jeon [16], assuming the training procedure trains decision trees with at most $k$ training points per leaf node, and the full training set is used for each tree, then any point $\mathbf{x}_i$ with positive weight is in fact a $k$-nearest neighbor under some monotone distance metric. Moreover, due to the training procedure of each constituent regression tree, this function assigns high weight to pairs of points that share a similar value of the response, $y$. Hence, we can interpret it as providing an adaptive distance measure, where distances are shortened in directions where the function $E(y \mid \mathbf{x})$ is flat. This is illustrated in Figure 2, where we plot the values of the weighting function for a random forest fit on a function of two variables that does not vary in the first coordinate, when making a prediction at the origin $(0, 0)$. Note that the weights taper off much more quickly along the $x_2$ dimension, and that the random forest 'borrows strength' from points that have very different values of $\mathbf{x}_1$.

## 3 SUPERVISED NEIGHBORHOODS

The prior discussion sets up the introduction of our method for nonparametric regression. The first step is to use the standard random forest training algorithm to fit a global random forest to the training data $\{(y_i, \mathbf{x}_i)\}$. This yields a random forest weighting function $w_{\mathrm{RF}}(\cdot, \cdot)$; note that, given the training set, this

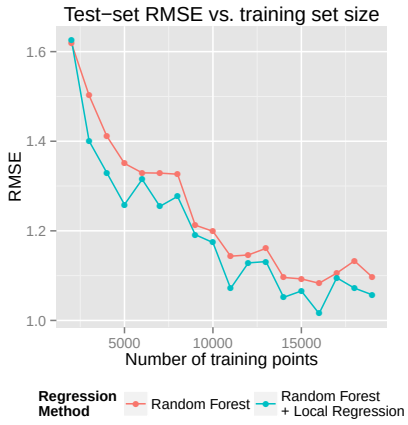


Figure 3: Test-set RMSE of standard random forest vs. random forest combined with local linear modeling.

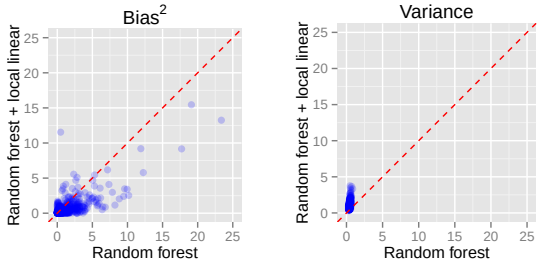


Figure 4: Empirical bias and variance of standard random forest vs. random forest + local linear regression, on the same prediction task as Figure 3, with 5000 training points.

is still a random function, as it depends on the underlying tree-training parameters $\theta_1, \ldots, \theta_K$. We then use this weighting function in equation 1, but unlike random forests, we expand the function class $\mathcal{F}$ to a broader, more flexible class than just constants. In our experiments and theoretical analysis, we use the set of linear functions with an intercept term:

$$\mathcal{F} = \left\{ f \text{ s.t. } f(\mathbf{x}) = \alpha + \boldsymbol{\beta}^T \mathbf{x}; \alpha \in \mathbb{R}, \boldsymbol{\beta} \in \mathbb{R}^p \right\}. \quad (7)$$

For convenience, define $w_i = w_{\text{RF}}(\mathbf{x}_i, \mathbf{x})$, and let $\mathbf{U}(\mathbf{x}) \in \mathbb{R}^{p+1} = (1, x_1, \ldots, x_p)^T$, i.e. $\mathbf{U}(\mathbf{x})$ is the vector $\mathbf{x}$ prefixed by 1. Substituting into equations 1 and 2, to make a prediction at a point $\mathbf{x} \in \mathbb{R}^p$, our method can be written in the following form:

$$\boldsymbol{\beta}_{\mathbf{x}} = \underset{\boldsymbol{\beta} \in \mathbb{R}^{p+1}}{\arg\min} \sum_{i=1}^{n} w_i \left( y_i - \boldsymbol{\beta}^T \mathbf{U}(\mathbf{x}_i - \mathbf{x}) \right)^2 \quad (8)$$

$$\hat{g}(\mathbf{x}) = \boldsymbol{\beta}_{\mathbf{x}}^T \mathbf{U}(\mathbf{0}). \quad (9)$$

Equation 8 is standard weighted linear regression, re-centered at $\mathbf{x}$, and the prediction at point $\mathbf{x}$ is the resulting intercept term (which motivates evaluating $\boldsymbol{\beta}_{\mathbf{x}}$ at $\mathbf{U}(\mathbf{0})$). We note that this method can, in fact, be written as a local averaging method, but where the original weights from the random forest are transformed [15]. Define

$$\boldsymbol{\Sigma}_{\mathbf{x}} = \sum_{i=1}^{n} w_i \mathbf{U}(\mathbf{x}_i - \mathbf{x}) \mathbf{U}(\mathbf{x}_i - \mathbf{x})^T \quad (10)$$

$$\mathbf{a}_{\mathbf{x}} = \sum_{i=1}^{n} \mathbf{U}(\mathbf{x}_i - \mathbf{x}) w_i y_i \quad (11)$$

Then, if we assume that $\boldsymbol{\Sigma}_{\mathbf{x}}$ is invertible,

$$\begin{aligned} \hat{g}(\mathbf{x}) &= \mathbf{U}(\mathbf{0})^T \boldsymbol{\Sigma}_{\mathbf{x}}^{-1} \mathbf{a}_{\mathbf{x}} \\ &= \frac{1}{n} \sum_{i=1}^{n} \mathbf{U}(\mathbf{0})^T \boldsymbol{\Sigma}_{\mathbf{x}}^{-1} \mathbf{U}(\mathbf{x}_i - \mathbf{x}) w_i y_i \end{aligned} \quad (12)$$

Thus, local modeling transforms the weights as $w_i \to W_i := \mathbf{U}(\mathbf{0})^T \boldsymbol{\Sigma}_{\mathbf{x}}^{-1} \mathbf{U}(\mathbf{x}_i - \mathbf{x}) w_i$. Note that the weight update is unsupervised ($y$ is not incorporated), and has the effect of correcting local imbalances in the design; for an empirical investigation of this property, see, for example, Hastie and Loader [17]. It is a well-known feature of local linear regression that it reproduces linear functions exactly; see proposition 1.12 of Tsybakov [15]. Hence the transformed weights balance the design about the point $\mathbf{x}$; we have that:

$$\frac{1}{n} \sum_{i=1}^{n} \mathbf{U}(\mathbf{0})^T \boldsymbol{\Sigma}_{\mathbf{x}}^{-1} \mathbf{U}(\mathbf{x}_i - \mathbf{x}) w_i \mathbf{U}(\mathbf{x}_i - \mathbf{x})^T = \mathbf{0}$$

Note that the term $\mathbf{U}(\mathbf{0})^T \Sigma_{\mathbf{x}}^{-1}$ extracts the first row of $\Sigma_{\mathbf{x}}^{-1}$; the elements of this row are related to the difference between $\mathbf{x}$ and the weighted average of the $\mathbf{x}_i$; we explictly calculate this vector in the proof of Theorem 1.

Figure 3 compares the predictive root mean-square-error (RMSE) of standard random forests with local-regression augmented random forests, i.e., SILO, when training on the Friedman1 simulated dataset, for varying sizes of training sets. We see that the additional local linear modeling step improves the predictive RMSE. In Figure 4 we calculate an empirical bias-variance trade-off for predictions of each method. To calculate these estimates of bias and variance, we fit the models 500 times, fixing all the training and test set data except the values of $\epsilon_i$ in the training data (i.e. this is bias and variance conditional on the design). We then estimate the bias by calculating the difference between the mean prediction over the 500

models and the true expected outcome, and the variance is estimated by calculating the variance of the predictions over the 500 models. We see that local modeling reduces the bias of the original random forest at the expense of a slight increase in the variance; overall the RMSE is diminished.

### 3.1 Consistency of random forests + local regression

Consistency properties of random forests have been extensively studied since the original results in [8]. The true random forest algorithm is notoriously difficult to analyze, due to the many complex ingredients (greedy partitioning scheme, bagging, and deeply trained trees), so most analyses make simplifications to make the analysis more tractable. Local polynomial methods based on Euclidean distance, on the other hand, are very well understood theoretically: they are known to have minimax-optimal rates of convergence [18], can correct bias problems at the boundary of the data [19], and that, with slight modifications to avoid pathological behavior, they are known to be consistent over arbitrary joint distributions of $y$ and $\mathbf{x}$ with $Ey^2 < \infty$ [20].

In our analyses, we make several simplifications to the true random forest algorithm. Most notably, our analysis relies on the assumption that the procedure used to build the regression trees proceeds independently of the data used in building the local models; this assumption is also made in [21] and is similar to the 'honest tree' assumption defined in [22]. The more detailed mathematical analysis in [23] proves consistency of random forests for additive regression functions while avoiding this assumption. In our results, we refer the 'training data' as the dataset $\{(y_i, \mathbf{x}_i)\}$ which is used to fit local models, but is not used to determine the structure of the trees. Also, we assume that these data are sampled without replacement in each of the trees of the forest, to avoid difficulties in analyzing sampling with replacement. More specifically, we require the following:

**Assumption 1.** *The training data $\{(y_i, \mathbf{x}_i), i = 1, \ldots, n\}$ are generated i.i.d. from a joint distribution that satisfies the following properties:*

$$\mathbf{x}_i \sim \text{Uniform}([0,1]^p) \tag{13}$$

$$y_i = g(\mathbf{x}_i) + \omega(\mathbf{x}_i)\epsilon_i \tag{14}$$

*where $\epsilon_i$ is independent of $\mathbf{x}_i$, the function $\omega(\mathbf{x})$ is bounded, $E(\epsilon_i) = 0$, and $E(\epsilon_i^2) < \infty$. The function $g$ must be sufficiently well-behaved such that Assumption 4 can be satisfied. A minimal requirement is that $g$ is continuous.*

**Assumption 2.** *The splits of the constituent regression trees of the random forest are calculated using a dataset that is independent of the training data.*

**Assumption 3.** *We require that*

$$\min_{\substack{\mathbf{x}\in[0,1]^p \\ j\in(1,\ldots,K)}} k_{\theta_j}(\mathbf{x}) \to \infty \tag{15}$$

*i.e., the number of training points contained in each node of each tree of the random forest goes to infinity.*

**Assumption 4.** *For each $\mathbf{x} \in [0,1]^p$, the trees are trained in such a way that*

$$\max_{i,j} [w(\mathbf{x}_i, \mathbf{x}, \theta_j)|g(\mathbf{x}_i) - g(\mathbf{x})|] \xrightarrow{p} 0 \tag{16}$$

*i.e., that the cells containing $\mathbf{x}$ shrink in such away that the maximal variation of the function $g$ within a cell shrinks to 0 in probability.*

**Assumption 5.** *The data in each tree are sampled without replacement from the original training set, so that all training points occuring in a particular leaf node have the same weight.*

**Theorem 1.** *Under Assumptions 1 - 5, for all $\mathbf{x} \in [0,1]^p$,*

$$\hat{g}(\mathbf{x}) - g(\mathbf{x}) \xrightarrow{p} 0 \tag{17}$$

The proof is provided in the supplementary material.

**Remark 1.** *Assumption 4 allows us to ensure that the approximation error at point $\mathbf{x}$ vanishes asymptotically. This can be shown to hold, for example, by combining Lemma 2 of Meinshausen [24] with continuity of $g$; however, this requires additional assumptions on the tree-training procedure that deviate somewhat from the usual random forest tree-training procedure. Proposition 2 in Scornet, Biau, and Vert [23] proves a slightly weaker version of this assumption: the pointwise statement is not proven, but a similar statement is shown to hold for a random draw of $\mathbf{x}$. It is an open question whether such pointwise control over the leaves can hold for the true random forest tree-training procedure.*

## 4 DISTRIBUTED Silo

There is a rich literature on algorithms for recovering nearest neighbors from large datasets in high dimension, leading to efficient implementations of $k$-nearest neighbors in the distributed setting [25]–[27]. The possibility of using supervised neighborhoods in a distributed setting remains less explored. We propose an approach for distributed nonparametric estimation which is based on the random forest-supervised neighborhood method introduced above. In the setting of distributed data, communication latency is

**Algorithm 1:** Distributed Nonparametric Regression via SILO

**Input:** Number of workers $W$; Distributed training dataset $\{(y_{i,j}, \mathbf{x}_{i,j}); j = 1, \ldots, W; i = 1, \ldots, N_j\}$ where $\sum_j^W N_j = n$; Test points $(\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_M)$ stored on master node

**for** *each worker* $j = 1 \to W$ **do in parallel**
  Fit random forest on local data $(y_{i,j}, \mathbf{x}_{i,j})$
**for** *each* $m = 1 \to M$ **do**
  Master broadcasts test point $\tilde{\mathbf{x}}_m$
  **for** *each worker* $j = 1 \to W$ **do in parallel**
    Worker $j$ calculates supervised neighborhood of $\tilde{\mathbf{x}}_m$, yielding $\{w_{i,j}, (y_{i,j}, \mathbf{x}_{i,j})\}$
    Worker $j$ calculates weighted covariance matrix and cross-covariance vector:

$$\Sigma_{j,m} = \sum_i^{N_j} w_{j,i} U\left(\mathbf{x}_{j,i} - \tilde{\mathbf{x}}_m\right) U\left(\mathbf{x}_{j,i} - \tilde{\mathbf{x}}_m\right)^T \qquad \mathbf{a}_{j,m} = \sum_i^{N_j} w_{j,i} U\left(\mathbf{x}_{j,i} - \tilde{\mathbf{x}}_m\right) y_{j,i}$$

    Worker $j$ communicates $(\Sigma_{j,m}, \mathbf{a}_{j,m})$ to the master
  Master node solves linear system and calculates $\hat{g}\left(\tilde{\mathbf{x}}_m\right) = U\left(\mathbf{0}\right)^T \left[\sum_{j=1}^W \Sigma_{j,m}\right]^{-1} \sum_{j=1}^W a_{j,m}$
**return** $\left(\hat{g}\left(\tilde{\mathbf{x}}_1\right), \ldots, \hat{g}\left(\tilde{\mathbf{x}}_M\right)\right)$

high, and implementing the random forest algorithm on a distributed dataset is prohibitively expensive, due to the necessity of performing several distributed sorts at each internal node in the CART training procedure. However, approximations based on binning data, extracting order statistics, and using randomized searches for split-points has led to several scalable implementations of distributed random forests [11], [28].

Here, we take a different approach, which avoids communication at training time. We extend SILO to the distributed setting, but instead of attempting to train a global random forest, we train random forests separately on each worker. At test-time, supervised neighborhoods of each test point are determined independently by each worker node, using the usual random forest procedure of finding weighted PNNs. Then the master node gathers the workers' supervised neighborhoods and fits a local linear model. The local model helps to lower the bias of the workers' individual predictions; this is similar in spirit to the approach taken in Zhang, Wainwright, and Duchi [29], which uses bootstrap bias correction. Our method is presented in more detail in Algorithm 1. Note that, instead of communicating the training data itself, the workers communicate sufficient statistics for performing local regression - the weighted covariance matrix $\Sigma_{\mathbf{x}}$ and the weighted cross covariance vector $a_{\mathbf{x}}$; the master then must simply add together these results from the workers and solve a linear system. Additionally, to amortize latency costs, we batch the communication at test-time; the master can broadcast a set of several test-points in a single message, and the workers can communicate a corresponding set of sufficient statistics. In practice, this significantly speeds up test-time computations.

### 4.1 Simulation and Real Data Experiments

We analyze the performance of Algorithm 1 in two ways. First, we plot the out-of-sample predictive accuracy of distributed estimation as we increase the number of worker nodes but fix the size of the overall dataset. Second, we plot the accuracy as the size of the dataset is fixed per worker, but the number of workers is varied. We compare the performance of our algorithm to both naive averaging of divide-and-conquer random forests (equivalently, local constant models in Algorithm 1), and, in the second case, the global distributed random forest implementation in MLlib. Runtimes are also shown in the supplemental materials.

#### 4.1.1 Fixing the training set size

We demonstrate the effect of parallelization on two simulated datasets and one real dataset. The first simulation is the Friedman1 function described in Section 2.1. The second simulation is a Gaussian process, generated by fixing $M$ different vectors in $\mathbb{R}^p$, drawing $Z_1, \ldots, Z_M$ i.i.d. $\mathcal{N}(0, 1)$, setting $\sigma^2 = 0.05$, and generating the function $g(\mathbf{x}) = \sum_{k=1}^M Z_k e^{-\frac{1}{2}\|\boldsymbol{\omega}_k\|^2 \sigma^2} \cos(\boldsymbol{\omega}_k^T \mathbf{x})$. We generate a function in $p = 10$ variables, but for our simulation, we append an additional 20 variables unrelated to the outcome. This prediction task is very difficult - even with 1 million training points, a random forest attains a test-set correlation of 0.83. The real dataset is a pre-

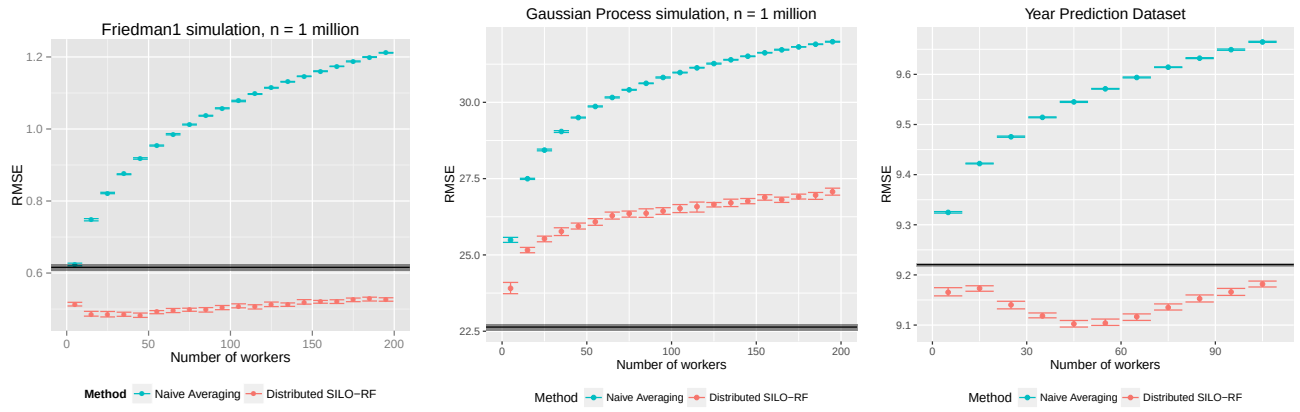


Figure 5: Effect of distributing the data for the Friedman1 and Gaussian Process simulations, and the Year Prediction dataset. The black bar represents a random forest trained on the full dataset. The error bars represent the ranges between the 0.1 - 0.9 quantiles after performing 40 different runs of the experiments, fixing the training and test data data, but fitting different random forests. In the case of the simulations, the test datasets are noise-free, so the true regression function would have an RMSE of zero.

diction task drawn from the Million Song Dataset [30]. The task is to predict the year of a song's release given 90 acoustic features of the song. The training set consists of 463,715 songs appearing between 1922 and 2011, and the test set consists of 51,630 songs. The results are shown in Figure 5. The plots show the performance of a random forest trained on the full datasets (black lines), a 'naive'-divide and conquer random forest, which simply averages predictions from random forests trained on the workers (blue points), and the distributed SILO procedure outlined in Algorithm 1. The number of workers ranges from 5 to 195 for the two simulations (corresponding to 200k to 5k training points per worker), and from 5 to 105 for the Year Prediction task (corresponding to approximately 92k to 4k training points per worker). The error bars in the plots represent 0.1-0.9 quantile ranges, as the experiments were repeated 40 times, holding the training and test set fixed, while fitting different random realizations of random forests. We see that, in the case of the Friedman1 simulation, the Distributed SILO procedure consistently outperforms the full random forest and, in contrast to the naively distributed random forest, shows little decay in performance as the data are distributed. In the Gaussian process simulation, the performance of Distributed SILO does deteriorate as the data are distributed, and we can see an increase in variance, but it still significantly outperforms the naively distributed random forest. For the Year Prediction task, Distributed SILO significantly improves upon the full random forest for all numbers of workers.

#### 4.1.2 Fixing the dataset size per worker

We now explore the performance of distributed-SILO when we fix the training dataset size per worker, but increase the number of workers, and hence, the overall amount of training data. We use the two simulation setups outlined above: the Friedman1 function, and the higher-dimensional Gaussian process. We slightly altered the Friedman1 simulation, adding an additional 45 noise features to increase the overall size of the training dataset and to make the random forest fitting procedure more computationally challenging, as the `mtry` parameter (number of variables considered at each node) is increased from 3 to 18. We implemented distributed-SILO in Spark, a popular open source framework for distributed computation, and we compare our method with the implementation of distributed random forests in MLlib. To attain similar accuracy between SILO and MLlib's random forests, we set the `maxDepth` parameter of MLlib to be 15 and the `minInstancesPerNode` parameter to be 10. We ran our experiments on Amazon EC2, using r3.xlarge instances, which have 4 processors and 30.5 GB of RAM per node. We set the number of Spark partitions to equal the number of processors, testing clusters of size 2 to 8 nodes. The training dataset size was fixed at n = 100,000 per partition, yielding $n$ ranging from 800k to 3.2 million. The results of our experiments are shown in Figure 6. In these plots we show three different methods: Distributed-SILO, naive divide-and-conquer random forests, and MLlib's distributed random forest. Similar to the results in Figure 5, the local modeling step in SILO improves performance relative to the naive averaging method for most cluster sizes. The performance of Distributed-SILO is particularly strong


Figure 6: Performance of SILO, MLlib's random forest, and naively distributed random forests on simulated datasets with growing training set size. We fixed the amount of training data per partition to 100,000 observations, and varied the number of partitions from 4 to 32. The y-axis shows the test-set RMSE. In the Friedman1 simulation, SILO consistently fits better models with more training data. In the Gaussian process simulation, both SILO and MLlib generally improve with more training data. For most cluster sizes, SILO improves upon naive averaging of workers' predictions. For large cluster sizes, it is more than 2× faster than MLlib due to its avoidance of communication; see the supplementary material for details.

in the Friedman1 simulation, as the predictions consistently improve with more training data, unlike MLlib. In the Gaussian process simulation, both Distributed-SILO and MLlib generally improve with more data, though the trend is not monotone. Training time of Distributed-SILO is essentially constant because it avoids communication entirely. In both simulations, it is more than 2× faster than MLlib for the largest clusters (see the supplemental material).

We note that distributed-SILO has traded training time for test time, as communication of supervised neighborhoods between workers and master must occur, and a local model must be fit for every test point. However, we find that the increase in test time is negligible compared to the gains in training time; for example, by batching test points into groups of size 100, we are able to amortize the cost of communication latency, and find that the predictions can be made at an overall rate of approximately 20 milliseconds per test point. While this is several orders of magnitude slower than making predictions using a model that is stored locally (as is the case with MLlib), it is still fast enough that overall gains in training time are dominant unless very large test sets are required. We also note that, in the standard random forest algorithm, the trees are built such that the leaves contain a small number of test points (usually less than 10 for regression tasks). Thus, the overall storage costs of random forests scales linearly with the size of the training set, and the models may grow beyond the in-memory storage capacity of a single machine; for example, we estimate that, in our Scala implementation, a random forest trained on a dataset of size 3.2 million points would require more than 6 gigabytes of memory.

## 5 CONCLUSION

SILO is a novel local learning algorithm that uses random forests to identify supervised neighborhoods for the problem of non-parametric regression. We proved the consistency of SILO, introduced a distributed variant, and demonstrated its favorable empirical performance (in terms of accuracy and computation) relative to natural baselines.

We note that the contemporaneous work of [31] introduces a local learning method that also relies on random forests to identify supervised neighborhoods for non-parametric regression. They also introduce a reweighting procedure for the local models that, in contrast to ours, is supervised using a small scale local random forest. This work focuses on empirical studies and does not investigate the scalability of the proposed algorithm. However, in followup work, they show that the underlying ideas motivating the distributed variant of SILO are applicable to their approach as well [32].

Moving forward, it would be interesting to extend SILO to the classification setting, study the degree to which SILO can be parallelized by characterizing the relationship between $n$, $N_j$ and $W$ in Algorithm 1, and investigate the theoretical performance of local methods with supervised neighborhoods relative to classical non-adaptive methods under sparsity assumptions, e.g., when the response only involves $s \ll p$ predictors.

**Acknowledgements** This work was partially supported by NSF grants DMS-1107000, CDS&E-MSS-1228246, DMS-1160319 (FRG), AFOSR grant FA9550-14-1-0016, The Center for Science of Information (CSoI), an US NSF Science and Technology Center under grant agreement CCF-0939370, and the National Defense Science & Engineering Graduate Fellowship Program (NDSEG). AT is supported in part by a Bloomberg Research Grant and an AWS in Education Research Grant.

## References

[1] B. Recht, C. Re, S. Wright, and F. Niu, "HOGWILD!: a lock-free approach to parallelizing stochastic gradient descent," in *NIPS*, 2011.

[2] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *NIPS*, 2010.

[3] J. Dean, G. Corrado, *et al.*, "Large scale distributed deep networks," in *NIPS*, 2012.

[4] M. Jaggi, V. Smith, *et al.*, "Communication-efficient distributed dual coordinate ascent," in *NIPS*, 2014.

[5] L. W. Mackey, M. I. Jordan, and A. Talwalkar, "Divide-and-conquer matrix factorization," in *NIPS*, 2011.

[6] T. Hastie and R. Tibshirani, "Discriminant adaptive nearest neighbor classification," *IEEE Trans. Pattern Anal. Mach. Intell. 18*(6): 607–616, 1996.

[7] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[8] L. Breiman, "Random forests," *Mach. Learn. 45*(1): 5–32, 2001.

[9] J. Lafferty and L. Wasserman, "Rodeo: sparse, greedy nonparametric regression," *Ann. Statist. 36*(1): 28–63, 2008.

[10] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proc. 2nd HotCloud*, 2010.

[11] X. Meng *et al.*, "Mllib: machine learning in apache spark," *ArXiv e-prints*, 2015. arXiv: 1505.06807 [cs.LG].

[12] W. S. Cleveland and S. J. Devlin, "Locally weighted regression: an approach to regression analysis by local fitting," *JASA 83*(403): 596–610, 1988.

[13] P. Chaudhuri, M.-C. Huang, W.-Y. Loh, and R. Yao, "Piecewise-polynomial regression trees," *Stat. Sin. 4*(1): 143–167, 1994.

[14] J. H. Friedman, "Multivariate adaptive regression splines," *Ann. Statist. 19*(1): 1–67, 1991.

[15] A. B. Tsybakov, *Introduction to Nonparametric Estimation*. Springer-Verlag New York, 2008.

[16] Y. Lin and Y. Jeon, "Random forests and adaptive nearest neighbors," *JASA 101*(474): 578–590, 2006.

[17] T. Hastie and C. Loader, "Local regression: automatic kernel carpentry," *Stat. Sci. 8*(2): 120–129, 1993.

[18] C. J. Stone, "Optimal rates of convergence for nonparametric estimators," *Ann. Statist. 8*(6): 1348–1360, 1980.

[19] J. Fan and I. Gijbels, "Variable bandwidth and local linear regression smoothers," *Ann. Statist. 20*(4): 2008–2036, 1992.

[20] M. Kohler, "Universal consistency of local polynomial kernel regression estimates," *Ann. Inst. Stat. Math. 54*(4): 879–899, 2002.

[21] G. Biau, "Analysis of a random forests model," *JMLR 13*: 1063–1095, 2012.

[22] S. Wager and S. Athey, "Estimation and inference of heterogeneous treatment effects using random forests," *ArXiv e-prints*, 2015. arXiv: 1510.04342 [stat.ME].

[23] E. Scornet, G. Biau, and J.-P. Vert, "Consistency of random forests," *Ann. Statist. 43*(4): 1716–1741, 2015.

[24] N. Meinshausen, "Quantile regression forests," *JMLR 7*: 983–999, 2006.

[25] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM 18*(9): 509–517, 1975.

[26] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proc. Thirtieth Annu. ACM Symp. Theory Comput.*, 1998.

[27] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Commun. ACM 51*(1): 117–122, 2008.

[28] B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo, "PLANET: massively parallel learning of tree ensembles with MapReduce," *Proc. VLDB Endow. 2*(2): 1426–1437, 2009.

[29] Y. Zhang, M. J. Wainwright, and J. C. Duchi, "Communication-efficient algorithms for statistical optimization," in *NIPS*, 2012.

[30] T. Bertin-Mahieux, D. P. W. Ellis, B. Whitman, and P. Lamere, "The million song dataset," in *Proc. 12th ISMIR*, 2011.

[31] R. Xu, D. Nettleton, and D. J. Nordman, "Case-specific random forests," *J. Comput. Graph. Stat. 25*(1): 49–65, 2016.

[32] J. Zimmerman and D. Nettleton, "Case-specific random forests for big data prediction," in *JSM Proceedings, Gen. Methodol.*, 2015.