Bag of Little Random Forests (BLRF)

December 17, 2017

Abstract

Random Forests (Breiman, 2001) are an ensemble method that utilizes a number of decision trees to make robust predictions in both regression and classification settings. However, the process of bootstrap aggregation, the mechanism underlying the Random Forests algorithm, requires each decision tree to physically store and perform computations on data sets of the same size as the training data set. This situation is oftentimes impractical given the large size of data sets nowadays. To address this problem, we introduce the Bag of Little Random Forests (BLRF), a new algorithm that combines the Random Forests with the Bag of Little Bootstraps (Kleiner et al., 2014) resampling method, aiming to achieve a better computational profile while producing predictions with comparable accuracy as those of the Random Forests.

1 Introduction

With the advent of the era of big data, machine learning techniques have become prevalent in solving problems across areas ranging from astrophysics to pharmaceutics. Equipped with powerful models, data scientists and business analysts have managed to make unprecedented advancement in areas like planetary science and bioinformatics (Mjolsness and DeCost, 2001).

Random Forests (Breiman, 2001), or RF, are among one of the top performing machine learning algorithms that is robust in numerous regression and classification settings (Caruana et al., 2008). Essential to the Random Forests algorithm is the idea of bootstrap aggregation, also known as "bagging"(Kleiner et al., 2014), which means the forests consist of a number of decision trees during training time and output the mode or the average results of all the trees as prediction. Bootstrap aggregation is powerful in that it mitigates the well-known variation and over-fitting problems of decision trees, yielding high accuracy (Gareth James and Tibshirani, 2013).

Despite their effectiveness, many machine learning algorithms require intense computational resources. In the case of Random Forests, bootstrapping from the original sample requires each decision tree to physically store and perform computations on resamples of the input training set, whose size can be massive. The challenge to address such a problem makes an alternative resampling method stand out, Bag of Little Bootstraps (Kleiner et al., 2014) .

Bag of Little Bootstraps, also known as BLB, is a resampling method devised by Kleiner et al. (2014). In their paper, Kleiner et al. showed that BLB can "yield a robust, computationally efficient means of assessing the quality of estimators" (Kleiner et al., 2014, page 1,2). BLB ingeniously combines subsampling and bootstrapping such that bootstrapping is performed on small subsets of distinct observations from the original sample. Because bootstrapping is only performed on each subset, the computational cost is favorably rescaled to the size of a subset. In addition, given that calculations for each subset are independent of one another, the BLB structure is well-suited to distributed and parallel computing architectures that are often used to achieve better computational performance.

In light of Kleiner's paper, we see the opportunity to use BLB to replace the process of bootstrapping in the standard Random Forests algorithm, aiming to achieve a better computational profile while producing predictions with comparable accuracy to those of Random Forest.

The following sections will be structured as follows: In Chapter 2, we will introduce the underlying algorithm of Random Forests, highlighting their sources of computational demands; in Chapter 3, we will give an overview of the method of Bag of Little Bootstraps and review its theoretical foundations laid out by Kleiner el al.; Chapter 4 integrates RF and BLB by introducing Bag of Little Random Forests (BLRF), covering its algorithm and main advantages over the standard Random Forests; then, we will run some simulations to assess BLRF's statistical and computational performance relative to the standard RF in Chapter 5; finally, we will conclude with some further discussions on possible modifications and future directions in Chapter 6.

2 Random Forests

As we mentioned in Chapter 1, the Random Forest (RF) algorithm is a powerful machine learning algorithm in both classification and regression settings. More importantly, RF is built on top of a fundamental building block, the Classification and Regression Tree (CART) algorithm (Leo Breiman, 1999). In this chapter, we will first cover how CART models are constructed and then move on to how they are transformed into Random Forests. Also, we will highlight the sources

of computational burden brought by bootstrap aggregation, a method we will explain in detail, to pave the way for the discussion of Bag of Little Random Forests in Chapter 4.

2.1 Classification and Regression Tree (CART)

As its name suggests, the CART algorithm, like Random Forests, can be applied to classification and regression problems. The trees are constructed by recursively partitioning the training observations into a number of $regions$, (or $nodes$ of a tree), that represent the most homogeneous space with respect to the response variable for the data. Since this paper is concerned with the regression setting, the following sections are dedicated to the fundamentals of CART regression trees.

2.1.1 Regression Tree

In algorithmic terms, a CART regression tree functions in the following way:

1. The CART algorithm is a greedy method used to partition the predictor space - the possible values for $X_1, X_2, ...X_p$ - into distinct and non-overlapping regions, $R_1, R_2, ..., R_J$. The goal is to find regions, or high-dimensional boxes $R_1, R_2, ..., R_J$, that minimize the Sum of Squared Errors (SSE) for the tree, defined as:

$$SSE_{tree} = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \bar{y}_{R_j})^2 \tag{1}$$

where $\bar{y}_{R_j}$ is the average of the response variable for observations in region R_j. The reasoning behind such a criteria is that a high SSE means that the observations in a region are spread apart in terms of their response values. Hence, the best regression tree minimizes the SSE of the data.

2. To predict the response value of a new test observation, the algorithm passes this observation down the tree model constructed to determine which region, R_j, the observation falls into. Its predicted value will be the average of the response variable in that region.

As an example, suppose from Step 1 we determined that the predictor space should be divided into two regions: R_1 and R_2, where the average of response variables in R_1 is 5 and the average of the response variables in R_2 is 25. Given a new observation $X = x$ with unknown response Y, if $x \in R_1$, the model predicts Y to be 5 for this test observation.

Next, we will move on to discuss how these regions are constructed.

2.1.2 Recursive Binary Splitting

The procedure most important and most relevant to the construction of BLRF (as we shall see in Chapter 4) is called recursive binary splitting.

Recursive binary splitting is a top-$down$ and $greedy$ approach: "$top - down$" because the splits begin at the top of the tree (where all observations belong to a single region)l the algorithm then successively splits each region of points into two branches further down the tree; "$greedy$" because at each split, the model looks through all predictors and all of its possible splits to determine the split that maximizes the reduction in sum of squares of the response variable.

More specifically, to perform recursive binary splitting on a given node, t, we go through each one of the predictors, $X_1, X_2, ...X_p$, and all possible cut-off values, s (gaps in between the sorted array of a particular predictor X_j), that splits the observations within that node into two distinct regions : $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ (Gareth James and Tibshirani, 2013). Define the SSE of the node being split (parent node) as SSE_t and the two branches after the split (daughter nodes) as SSE_{t_L} and SSE_{t_R}. From all the possibilities, we choose the one that maximize the $reduction$ in SSE, defined by:

$$\Delta SSE = SSE_t - (SSE_{t_R} + SSE_{t_L}) \tag{2}$$

In a regression setting, the CART algorithm will continue such binary splitting procedure until a user-defined stopping condition is met (e.g., when the number of observations in a node is fewer than 5). The terminating condition is a crucial step in the CART algorithm, and this part will be revisited in Section 4.1.4. The resulting tree is one whose terminal nodes are closely centered around the average of their respective nodes.

It is worth noting that, to reduce the amount of computation, the SSE for a given node, R_t, with node size n_t can be calculated as the follows (Note that $SUM_t = \sum_{i \in R_t} y_i$, where y_i refers to the individual response variables within the node):

$$\begin{aligned}
SSE_t &= \sum_{i \in R_t} (y_i - \bar{y_t})^2 \\
&= \sum_{i \in R_t} (y_i^2 - 2 \cdot y_i \cdot \bar{y_t} + \bar{y_t}^2) \\
&= \sum_{i \in R_t} y_i^2 - 2 \cdot \sum_{i \in R_t} y_i \cdot \frac{SUM_t}{n_t} + \sum_{i \in R_t} \left(\frac{SUM_t}{n_t} \right)^2 \\
&= \sum_{i \in R_t} y_i^2 - 2 \cdot \frac{SUM_t^2}{n_t} + n_t \cdot \left(\frac{SUM_t}{n_t} \right)^2 \\
&= \sum_{i \in R_t} y_i^2 - 2 \cdot \frac{SUM_t^2}{n_t} + \frac{SUM_t^2}{n_t} \\
&= \sum_{i \in R_t} y_i^2 - \frac{SUM_t^2}{n_t}
\end{aligned} \tag{3}$$

With this, Equation (2) can be rewritten as:

$$\begin{aligned}
\Delta SSE &= SSE_t - (SSE_{t_R} + SSE_{t_L}) \\
&= \sum_{i \in R_t} y_i^2 - \frac{SUM_t^2}{n_t} - \left(\sum_{i \in R_{t_L}} y_i^2 - \frac{SUM_{t_L}^2}{n_{t_L}} + \sum_{i \in R_{t_R}} y_i^2 - \frac{SUM_{t_R}^2}{n_{t_R}} \right) \\
&= \frac{SUM_{t_L}^2}{n_{t_L}} + \frac{SUM_{t_R}^2}{n_{t_R}} - \frac{SUM_t^2}{n_t}
\end{aligned} \tag{4}$$

where R_{t_L} and R_{t_R} refer to the potential daughter nodes of node R_t, whose node sizes are n_{t_L} and n_{t_R} respectively.

Therefore, to find ΔSSE, we simply need to calculate the difference in $\frac{SUM^2}{n}$ before and after the split. This fact is crucial and useful in constructing the Bag of Little Random Forests algorithm in in a computationally efficient way.

2.1.3 Prediction

After the tree is built using the training data, we now have j distinct terminal regions (or terminal nodes), $R_1, R_2, ..., R_j$, corresponding to the predictor space. Given a new test observation, we can effectively "categorize" this observation by following down the branches corresponding to the predictor space of the test observation. The predicted response value for this test observation will be the average of the response variables within the terminal node to which it belongs (using the mean as the predicted value is specific to the regression setting).

2.2 Building a Random Forest

Although the CART algorithm seems straightforward and easy to understand, it suffers from two practical difficulties: high variability (slightly different training sets from the same population tend to produce completely different trees) and overfitting (the decision tree is inclined to make more-than-necessary splits that reduce training error at the cost of increasing testing error).

The algorithm of RF is proposed to address the problems of CART by adding two new "sources of randomness": bagging and random selection of variables.

2.2.1 Bagging

Bagging, also known as bootstrap aggregation, is a machine learning ensemble meta-algorithm designed to improve stability and accuracy of a set of weaker models. In the Random Forest setting, the idea behind bagging is to aggregate the results of multiple bootstrapped decision trees through averaging or majority vote.

More specifically, bootstrap aggregation is built on top of bootstrap (Efron and Tibshirani, 1994), a resampling method that measures the variability and accuracy (e.g., defined in terms of bias, variance, confidence intervals, or prediction error) of a statistic, $\hat{\theta_n}$[1], by approximating its true sampling distribution. We repeatedly sample with replacement up to size n from the given training observations to build a number of decision trees. The combined effect is that variability of the predictions is drastically reduced since the final result is given by the average prediction for all trees.

2.2.2 Random Forests

On top of bootstrap aggregation, there is one more step towards Random Forests: random selection of variables at each split for each decision tree.

Continuing from the bootstrapped decision trees, the Random Forests algorithm further "de-correlates" individual trees by making only a subset of all candidate predictors available at each split in the construction of an individual tree. This procedure effectively de-correlates the trees because if all the explanatory variables are available to all decision trees, it is very likely that most of the trees decide to split on several dominating variables and thus become similar to one another. When giving each tree only m out of p (often $m = \sqrt{p}$) predictors at each split, such a dominant effect would no longer exist, making the decision trees de-correlated. The main benefit of such de-correlation is the variance of decision trees is reduced since "averaging many highly correlated quantities (Bagging alone) does not lead to as large of a reduction in variance as averaging many uncorrelated quantities" (Gareth James and Tibshirani, 2013).

[1] n refers to the sample size.

2.2.3 Limitations of Random Forests

Even though Random Forests effectively mitigate the drawbacks of CART, they also suffer from some limitations, notably in memory consumption and computational time.

In terms of memory consumption, as we have seen in Section 2.2.1, Random Forests require a large number of (e.g., 500) decision trees whose sizes are the $same$ as the original training data set. Such setting can be troublesome because the training data set alone could be too massive to be stored in memory, not to mention RF builds hundreds of trees using data sets similar in size.

In addition, also due to the large size of the resamples, performing computations on even a single resample can be computationally demanding and time-consuming, not to mention that such computation have to be performed for hundreds of times.

Therefore, we are motivated to develop a modification to Random Forests that drastically reduces the computational burden while still maintaining the appealing accuracy of the original Random Forests. In light of such a need, we will continue our discussion with an alternative resampling/aggregation method called Bag of Little Bootstraps (BLB) in the following chapter, and how it can be integrated with Random Forests to the new algorithm, Bag of Little Random Forests in Chapter 4.

3 Bags of Little Bootstraps (BLB)

As discussed in Section 2.2.1, the bootstrap provides a simple and powerful means of assessing the quality of estimators (Efron and Tibshirani, 1994). However, because each resample is of the same size as the original data set, with roughly 63 percent of the observations present (Gareth James and Tibshirani, 2013), performing computation for even a single point estimate on large data sets presents computational challenges. Here, one might naturally turn to parallel and distributed computing structures, where each resample will be passed to a different core to process. But this method is still problematic since the large size of the bootstrap resamples renders "the cost of transferring data to independent processors or compute nodes ... overly high, as is the cost of operating on even a single resample using an independent set of computing resources" (Kleiner et al., 2014).

To mitigate such problem, Kleiner et al. introduced a new resampling method, Bag of Little Bootstraps (BLB). BLB combines features of the bootstrap and subsampling to form a resampling method well-suited for computations on large data sets while maintaining the favorable statistical properties of the bootstrap (Kleiner et al., 2014). In this chapter, we will briefly talk about BLB's notation, procedures and advantages. For a more rigorous discussion on BLB's statistical properties (consistency and higher-order correctness), refer to Kleiner et al. (2014).

3.1 Setting and Notation

Assume we have a set of training observations, $a_1, a_2, ..., a_n$, drawn from an unknown population P. We denote its empirical distribution as P_n. Using the training data, we are interested in computing an estimate $\hat{\theta}_n$ of some population parameter $\theta \in \Theta$. Here we use $Q_n(P)$ to represent the true sampling distribution of $\hat{\theta}_n$. We will also use $\hat{\theta}(P_n)$ to indicate that we used data from distribution P_n to compute the estimate, $\hat{\theta}_n$.

Recall from Section 2.2.1, the bootstrap aims to estimate the true sampling distribution of θ, $Q_n(P)$, from which we can compute a quality assessment, $\xi(Q_n(P))$ (e.g., confidence interval, standard error, or bias). Given a training sample with emperical distribution P_n, the bootstrap

method computes the approximation of $Q_n(P)$ by taking b resamples of size n with replacement from P_n; using the r^{th} resample, whose empirical distribution is $P_{n,r}$, the method calculates the bootstrap statistic $\hat{\theta}^*_{n,r}$ that approximates a realization of $\hat{\theta}_n$ from the population; then, with a vector of $\hat{\theta}^*_{n,r}$, the method proceeds by forming the empirical distribution of $\hat{\theta}^*$, denoted by Q^*_n; finally, it completes the approximation $\hat{\xi}(Q_n(P) \approx \xi(Q^*_n)$ by calculating the desired quality assessment using Q^*_n.

For the method of BLB, the essential goal and mechanism is the same, but we would like to break up the data into several subsamples, so that calculation on each sample can be processed in parallel.

3.2 Bags of Little Bootstraps (BLB)

3.2.1 Mechanism

BLB essentially functions by combining the results of performing the bootstrap on multiple small subsets of the incoming training sample. Before we proceed, it is useful to define the parameters that will be used in the procedure of BLB:

Parameter	Definition
n	Size of the training set
γ	The user-defined parameter that determines the size a each subsample
s	Total number of subsamples
b	Size of each subsample, $b = n^\gamma$

Table 1: Parameters of interest

First, the user picks the value of $\gamma \in [0, 1]$ and compute the value of b by $b = n^\gamma$. Then, we repeatedly take s subsamples of size $b < n$ by sampling without replacement from the given training set. In this way, the observations within each subsample are distinct (no repeated observations), whereas each observation may be present in multiple subsamples[2]. We denote the subsamples as $I_1, I_2, ..., I_s$ and using the notations in Section 3.1, the empirical distribution associated with subsample I_i is $Q^*_{n,i}$. After this, we bootstrap each subsample from **size *b* to size *n*** (so that each resample is still of size n), to obtain the quality assessment $\xi(Q^*_{n,i})$ for subsample I_i; finally, the BLB's estimate of $\xi(Q_n(P))$ is given by:

$$\hat{\xi}(Q_n(P)) = \frac{1}{s}\sum_{i=1}^{s} \xi(Q^*_{n,i}) \tag{5}$$

In the next section, we will discuss in detail how the resamples can be drawn in an effective way.

3.2.2 The Multinomial Method

One important aspect of the bootstrap is that it requires each resample to be of size n. As we have discussed at the end of section 3.2.1, in order for BLB to have consistent statistical performance, it also requires each resample to be of size n. Such setting, as we have mentioned, can be

[2]It is worth noting that, in Kleiner et al's paper, there is also the option of using s distinct and non-overlapping subsamples obtained by directly partitioning the training set. In such a setting, the subsamples are independent from one another, which could be useful in some situations.

troublesome because a data set of size n could be too big to store or compute. Fortunately, the multinomial method Kleiner et al. (2014) can be employed to scale each subsample of size b to n using only $\mathbf{O}(b)$ space.

The multinomial method makes use of the multinomial distribution, a generalized version of binomial distribution. Given a number n and a vector of probabilities associated with each outcome level $\mathbf{p} = \{p_1, p_2, ..., p_b\}$, the multinomial distribution can generate a vector of multinomial coefficients $\mathbf{M} = \{M_1, M_2, ..., M_b\}$ using the facts $P(M_i) = p_i$ and $\sum_{i=1}^{b} M_i = n$. In the BLB setting, n represents the size of the given training data set and to construct a resample of size n, $\mathbf{p} = \{1/b, 1/b, ..., 1/b\}$ since each observation should have equal probability of being chosen (akin to the bootstrap). In the resulting vector $\mathbf{M}$, each M_i coefficient represents the number of times that observation a_i is present in the resample of a subsample.

To better illustrate the multinomial method, figure 1 (Yenny Zhang, 2017) shows an example of a resample from subsample of size $b = 5$ up to size $n = 100$. In this case, the multinomial method produces the vector of multinomial coefficients $\mathbf{M} = \{21, 15, 22, 23, 19\}$, each representing the number of times an observation is present in this resample.

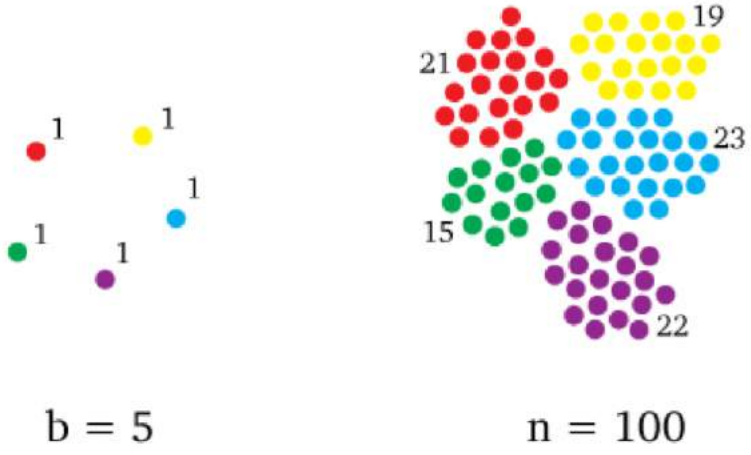

Figure 1: A Resample of one Subsample of the Training Observations

The complete algorithm of BLB, using confidence interval (CI) estimation as an example, is given in Algorithm 1 below.

Algorithm 1 Bag of Little Bootstraps (BLB)

INPUT:

$a_1, a_2, ..., a_n$, observations,	s, number of subsamples
$\hat{\theta}$, estimator of interest	r, number of resamples
b, size of subsamples	ξ, estimator quality assessment (e.g., CI)

OUTPUT: $\hat{\xi}(Q_n(P))$, the desired quality assessment

function BLB($A = \{a_1, a_2, ..., a_n\}$, s, r, b)
  for i in 1:s **do**
    \\ Subsample the data
    $I_i \leftarrow$ subsample of size b from A ▷ Subsample without replacement/Partition
    for j in 1:r **do**
      \\ Resample from the subsample
      $\mathbf{M}_j \leftarrow \{M_1, ..., M_b,\ \sum_{ind=1}^{b} M_{ind} = n\}$ ▷ the multinomial coefficients
      $P_{n,j} \leftarrow \mathbf{M}_j\ \times\ I_i$ ▷ replicate each observation by its coefficient
      $\hat{\theta}^*_{n,j} \leftarrow \hat{\theta}(P_{n,j})$ ▷ Estimate of the statistic of interest
    end for
    \\ Construct the CI for the statistic in subsample I_i
    $\xi(Q^*_{n,i}) \leftarrow \xi(\{\hat{\theta}^*_{n,1},\ ..., \hat{\theta}^*_{n,r}\})$
  end for
  \\ Average all the CIs obtained from s subsamples
  $\hat{\xi}(Q_n(P)) \leftarrow \frac{1}{s}\sum_{i=1}^{s} \xi(Q^*_{n,i})$
  return $\hat{\xi}(Q_n(P))$
end function

3.3 Advantages of the BLB Method

Compared to the standard bootstrap, the advantages of the BLB method are three-fold: computational, memory-wise, and architectural.

Computationally speaking, since each subsample is of size b, which in practice is much smaller[3] than n , the computation associated with BLB scales in $\mathbf{O}(b)$, as opposed to $\mathbf{O}(n)$ (Kleiner et al., 2014).

In terms of memory usage, the memory required by BLB is much smaller than that required by bootstrapping on the original data directly. Such reduction in memory usage results from the benefits of the mulitnomial method, which guarantees a $\mathbf{O}(b)$ memory consumption of each resample from the subsamples; the memory consumption of each resample from the standard bootstrap is of order $\mathbf{O}(n)$. However, it is important to note that, the BLB method requires several subsamples (the number of subsamples required depends on the value of γ) to obtain comparable results with those of the standard bootstrap. But as Kleiner et al. (2014) mentioned, for example, in the case of estimating the average, when $\gamma = 0.9$, we need only $s = 2$ subsamples to obtain a result comparable to that of bootstrap. In such a situation, if n is large, the combined memory usage of BLB is much smaller than that of the bootstrap[4]. On top of this, sometimes the data set is too large to

[3] To give readers a sense of the magnitude of b versus n, if $n = 10000$ and as γ ranges from 0.5 to 0.9, b ranges from $10000^{0.5} = 100$ to $10000^{0.9} = 3981$, substantially smaller than n.

[4] Assuming the number of resamples, r, is constant, when $n = 10000$ and $\gamma = 0.9$, BLB requires $r \times 10000^{0.9} = 7962r$ units of memory, whereas the bootstrap requires $10000r$ units of memory.

be handled by the bootstrap at all; in contrast, the BLB method makes such computation due to its reduced memory consumption.

Third, with its setup, BLB is more suited than the bootstrap to utilize the paralleled and distributed computing architecture. To perform BLB on a data set, we can pass each subample to an invidual compute node to complete its part of the calculation. One could argue that the original bootstrap can also utilize such a paralleled structure. But again, the massive size of the training observations might be too big for any single compute node to process. However, the BLB method uses $b < n$ points and thus can make the computation feasible while reducing computation time drastically.

4 Bag of Little Random Forests (BLRF)

With the theoretical background of Random Forest and Bag of Little Bootstraps, now we are ready to proceed to the introduction and discussion on Bag of Little Random Forests (BLRF). BLRF is a new machine learning algorithm that we developed based on the algorithm of RF with one major difference: instead of using bootstrap aggregation (or bagging), we will use "BLB Aggregation" to build multiple forests each built from many "BLB decision trees". In brief, there are two steps to the BLB Aggregation method: first, pick a value for γ, calculate $b = n^\gamma$ and repeatedly draw distinct subsamples of size b from the given training observations; second, build a "Little Random Forest" (LRF) with the observations from a subsample using the multinomial method; finally, we combine the results of the LRFs.

The desired outcome of BLRF is that we will be able to not only reduce computation time, but also produce results (measured in Relative Mean Squared Error (RelMSE) which will be further discussed in Chapter 5) comparable to that of the standard Random Forests algorithm.

In this chapter, we will go from "macro" to the "micro": first we will talk about the construction of the BLRF and the relationship in between the LRFs; then, we will delve into the necessary modifications to the the stopping criteria of each decision tree under the standard Random Forest algorithm (in a regression setting) and how it can work with the multinomial method to build a "Little Random Forest"; after that, we will discuss some advantages to the new BLRF algorithm; we will conclude with a higher order analogy of BLRF to Random Forests as Random Forests to CART.

4.1 BLB Aggregation

As its name suggests, "Bag of Little Random Forests" consist of a number of "Little Random Forests" and the final result is the aggregation of all the LRFs. More specifically, in a regression setting, each LRF is trained using the multinomial method on a subsample of training observations, while the final result given by the BLRF algorithm is the average of results produced by the LRFs. We call the above method "BLB Aggregation".

Before we proceed, it is useful to draw the connection between parameters for the BLB method and parameters used in BLB Aggregation:

Note that all of the symbols are kept the same except r, which is now denoted as $ntree$. Since each subsample corresponds to a single LRF, the corresponding parameter of r, representing number of resamples of each subsample, should be $ntree$, representing number of decision trees within each LRF.

Parameter	Definition in BLRF
n	Size of the training set
γ	The user-defined sizing factor that determines value of b
s	Total number of Little Random Forests
b	Number of distinct observations in each Little Forest, $b = n^\gamma$
ntree (r)	Number of trees within each Little Forest

Table 2: Parameters in BLRF

4.1.1 Construction of the Bag - Subsampling

First, it is important to understand how subsmpling is used to construct the Bag and the relationship in between the Little Random Forests.

Given a set of training observations, $X_1, X_2, ..., X_n$, of size n, we repeatedly subsample, s subsets with distinct elements of size b from the training observations, where $b = n^\gamma$ and $\gamma \in [0, 1]$ is a user-defined parameter[5]. Note that the word "distinct" refers to the fact that, within each subsample, the observations have no overlap. But it is totally possible for a observation to be present in multiple subsamples, which means the LRFs are not completely independent[6].

Now, with a Bag of s subsamples of size b, we are ready to use each subsample to build a Little Random Forest.

4.1.2 Building a Little Random Forest (LRF)

Much like the standard Random Forests algorithm, the building block of a LRF is still a decision tree. However, there are two fundamental differences between a "Random Forest" and a "Little Random Forest": first, the Random Forest has access to all the available training observations, whereas each "Little Random Forest" only works on a subsample of size b; in addition, the Random Forest passes on a bootstrapped resample of size n, in which roughly 63% of all training observations are present, to each decision tree to perform its calculation, whereas a "Little Random Forest" passes b observations plus a vector of b multinomial coefficients, $\mathbf{M}$, to each decision tree, avoiding the potential problems caused by too large a resample.

More specifically, in a LRF, given the values of n and b, we will repeatedly construct the multinomial coefficient vector by using $\mathbf{M} = Multinom(n, \mathbf{p})$, where $\mathbf{p} = \{1/b, 1/b, ..., 1.b\}$. Then, we will pass the multinomial coefficient vector along with the b training observations assigned to the LRF to a number of (usually 500) decision trees. With that, we can delve into how a "BLB" decision tree is constructed within a LRF.

4.1.3 A "BLB" Decision Tree

As discussed in Section 2.1.2, the crux of the CART algorithm is the process of recursive binary splitting. Recall that, for each split, the algorithm goes through all possible splits - that is, all the "gaps" within all explanatory variables to find the $best$ split specified by Equation 4.

[5] For the predictions to be sufficiently accurate, Kleiner et al. (2014) determined that the optimal value of γ ranges from 0.7 to 0.9. This will be further discussed with illustration in Chapter 5.

[6] Like BLB, as discussed in Section 3.2.1, one also has the option of making the subsets independent by partitioning the training observations into non-overlapping subsets. The only drawback of such a method is that when n is small, say 500, and $\gamma = 0.9$, the value of $b = n\gamma = 500^{0.9} = 268$, the data can form only one subsample, but we need two or more subsamples to obtain a decent result.

Now, with the addition of the multinomial coefficients, $\mathbf{M}$, we modify the calculation for recursive binary splitting. Similar to the standard Random Forest, a LRF still de-correlates the trees by assigning only a subset of all predictors to each tree. However, remember that, instead of a bootstrapped resample, the information used to build a BLB decision tree is: the b training observations assigned to the LRF and the multinomial coefficients vector, $\mathbf{M}$. Thus, the algorithm now goes through the subset of explanatory variable and all gaps in between b observations. Also, similar to a decision tree, the criteria for finding the best split still uses Equation 4 (reproduced below as Equation 6), though with the addition of the multinomial coefficients, the calculations of the node sum, SUM, and the node population, n, are different:

$$\begin{aligned} \Delta SSE &= SSE_t - (SSE_{t_R} + SSE_{t_L}) \\ &= \frac{SUM_{t_L}^2}{n_{t_L}} + \frac{SUM_{t_R}^2}{n_{t_R}} - \frac{SUM_t^2}{n_t} \end{aligned} \tag{6}$$

and the new SUM and n are calculated as follows:

$$SUM = \sum_{i=1}^{b} y_i * M_i \tag{7}$$

$$n = \sum_{i=1}^{b} M_i \tag{8}$$

where y_i is the value of the response variable of observation i and M_i is the coefficient associated with that observation.

4.1.4 Terminating Condition for BLRF

Aside from the new way to calculate sum of a given node and its population, another important change that is made to the BLB decision tree relates to the terminating condition, i.e., when a node should not be further split. In the original CART, a commonly used criteria is stop splitting until the node population is smaller than 5. Now with the addition of the multinomial coefficients, the terminating condition is still the same, but note that the new population of a given node is calculated by Equation 8 - that is, stop the split if the sum of the multinomial coefficients associated with the observations within a given node is less than 5.

With all the theoretical background laid out, below we give the complete algorithm of BLRF:

Algorithm 2 Bag of Little Random Forests (BLRF)

INPUT:

$a_1, a_2, ..., a_n$, training observations — s, number of LRFs

γ, the user-defined sizing factor — $ntree$, number of trees within each LRF

OUTPUT: A BLRF object able to make predictions on new data

function BLRF($A = \{a_1, a_2, ..., a_n\}$, s, γ, $ntree$)
  $b \leftarrow n^{\gamma}$ ▷ b: size of each LRF
  \\ Build s Little Random Forests
  for i in 1:s **do**
    $I_i \leftarrow$ subsample of b distinct obs. from A
    \\ Use I_i to build a LRF
    for j in 1:$ntree$ **do**
      $\mathbf{M}_j \leftarrow \{M_1, ..., M_b, \ \sum_{ind=1}^{b} M_{ind} = n\}$ ▷ the Multinomial coefficients
      \\ Build a BLB decision tree using $\mathbf{M}_j$ and I_i
      $Tree_{i,j} \leftarrow$ BLB decision tree{$\mathbf{M}_j$, I_i}
    end for
    $LRF_i \leftarrow Aggregate_{j=1}^{ntree} \ Tree_{i,j}$
  end for
  $BLRF \leftarrow Aggregate_{i=1}^{s} \ LRF_i$
  return $BLRF$, the trained BLRF object
end function

4.2 Advantages

Much like the advantages of the BLB method as discussed in Section 3.3, the benefits of BLB Aggregation includes computational, memory-wise, and architectural gains.

First, BLB Aggregation drastically reduces the amount of computation, mainly due to the fact that each LRF only handles b distinct observations. More specifically, the computational gain mainly comes from the process of recursive binary splitting. Previously, a single tree inside the Random Forest has to process a resample of size n and thus it has to calculate Equation 6 for $0.632n$ times[7]. However, inside a LRF, each decision tree only needs to handle b observations and a vector of multinomial coefficients of size b. Now the number of time that we perform Equation 6 is strictly less than $b-1$, which in practice is much smaller than $0.632n$ [8]. Here, one may validly argue that, Bag of Little Random Forests needs to build a number of LRFs, not just one, and therefore the amount of computation may exceed that of the original Random Forest. Such issue is perfectly addressed by the paralleled structure of BLRF, which will be further discussed below.

In terms of memory usage, again due to the much smaller size of observations each LRF deals with, the amount of memory required for each LRF is much smaller than the original Random Forest. What is more, the algorithm and set up of BLRF makes computation and prediction possible on data sets that are otherwise too massive for the original Random Forest to work with. Facing a data set too large to be stored in memory at once, the original Random Forest cannot perform

[7] This is because, to avoid unnecessary computation, the gaps, or splits, in between repeated observations are skipped. Since each resample inside the original Random Forest roughly contains $0.632n$ distinct observations, the number of times that we perform Equation 6 is approximately equal to $0.632n$.

[8] As an example, for $n = 10000$ and $\gamma = 0.9$, $b = n^{\gamma} = 3981$, whereas $0.632n = 6320$.

any computation or give predictions at all, whereas BLRF is able to handle such a data set since the combined memory usage is much smaller than that of RF; and if even a subset is too big, we can pass each subset to an individual computing unit to build the model successfully.

Another major benefit that BLRF has over Random Forests is the computation of BLRF is more suited for the distributed and paralleled structure. As discussed in Section 3.3, we can pass each subsample, or a LRF, to an individual core and aggregate the results after the computation is done. If we have a multi-core computer, we can pass each of the LRF to an individual core to compute; if we have the computational resources of several multi-core computers, we can distributed each LRF to each computer wherein the resamples, or BLB decision trees, can be built using the computing power of a number of cores. In this way, the distributed and paralleled structure alleviates the computational burden brought by the need to construct a number of LRFs by utilizing the power of several computing cores and even several computers.

The above three advantages boasted by the BLRF algorithm makes it a compelling and practical algorithm well-suited for making predictions with massive data sets that are hard to process by the Random Forest.

4.3 Higher Order Analogy

Some readers may have noticed that, Random Forest to CART is as BLRF to Random Forest. The bootstrap aggregation method of Random Forest helps stabilize the variability of a decision tree by aggregating a large number of decision trees; BLRF uses BLB Aggregation and the predictability of several LRFs to enable predictions using massive data sets that cannot be processed by the Random Forest.

Now we have understood much of the method of BLRF. However, the theoretical set up does not give us much insight into the statistical and computational performance of the BLRF algorithm. Hence, in the next chapter, we will use the BLRF method on several data sets and assess its capabilities.

4.4 BLRF implementation in C

It is worth noting that, during the research, I implemented the regression model of BLRF on top of the original randomForest (Liaw and Wiener, 2002) package in R. The main modifications I made include: writing a function that generates random values from a multinomial distribution, passing the vector of multinomial coefficients and a subsample of $b = n^{\gamma}$ of observations to a LRF, changing the calculation of ΔSSE in recursive binary splitting according to equation 4, changing the R wrapper function to take in s and γ as parameters for BLRF, among other small changes. All modifications were written in the language of C.

5 Model Performance

To assess the statistical performance of the BLRF algorithm, we use BLRF to make predictions with simulated data and then compare its performance with predictions made by the standard Random Forest. The use of simulated data allows for the control of sample size and knowledge of the underlying relationship between the response and explanatory variables. In addition, to construct the Relative Mean Squared Error ($RelMSE$) mentioned at the beginning of Chapter 4, we need to build the standard Random Forest using the simulated data so that results of BLRF can be directly compared across different data structures.

In the following sections, we will briefly talk about the simulation set up, both the hardware and data structures, and then move on to the results of several simulation studies. With our results, we aim to show the effects of some important parameters of the BLRF algorithm - s, $ntree$ and γ - and that BLRF is a practical and asymptotically accurate algorithm. Here, "asymptotically" refers to when $\gamma -> 1$, or in other words, when each LRF essentially becomes a RF.

5.1 Simulation Setup

To evaluation the relative statistical performance of the BLRF algorithm, we use data generated from three data structures (see details below) and normalized the results by those obtained from the standard Random Forest algorithm. Since we are working exclusively within a regression setting, the normalized results are measured in $RelMSE$:

$$RelMSE = \frac{MSE_{BLRF}}{MSE_{RF}} \tag{9}$$

As we can see, "RelMSE" is defined by the MSE achieved by the BLRF under certain parameter settings (MSE_{BLRF}) divided by the optimal [9] MSE achieve by the Random Forest algorithm (MSE_{RF}). Thus, a RelMSE smaller than[10], equal to or slightly bigger than 1 is considered a good result (or "comparable to that of Random Forests").

All tests are run on three types of data structures: $linear$, $clustered$ and $cosine$. Each data set contains $n = 10000$ observations with $ndim = 5$ dimensions. For each realization of each data set, we first draw features $X \sim U([0,1]^5)$ and then generate the response variables, Y, using the following rules:

$linear$: $Y = 5X_1 + 10X_2 + 15X_3 + 20X4 + 25X_5 + \varepsilon$
$consine$: $Y = 50 \times cos(\pi \times (X_1 + X_2)) + \varepsilon$
$clustered$ [11]: $Y = cluster.mean + \varepsilon$

where ε is an error term generated from the standard normal distribution. Aside from the training observations, we also independently generate 2000 test observations from each data structures to assess the test accuracy of the models.

As for the hardware setup, all simulations are performed on a computer with 64 AMD Opteron 6276 CPUs running at 1.4 GHz. To reduce computation time, each of the tasks is performed in parallel using 20 cores. It is important to register more number of cores than number of LRFs, s, since each LRF should be built independently using a single core. In our simulation, s is at most 18.

5.2 Effect of Hyperparameters on Performance

With an overall understanding of the simulation setup, we are ready to discuss the simulation results. We will assess the simulation results through discussions on the three hyperparameters

[9] We define "optimal" as the average MSE of the Random Forest algorithm with 10 iterations and $ntree$ = 500.

[10] In practice, the probability of RelMSE being less than 1 is minimal. In other words, the performance of BLRF should be worse than those of RF, as the performance of BLB is relatively worse than that of the standard bootstrap (Kleiner et al., 2014).

[11] This data set is generated using the "simdataset" function under the MixSim package (Melnykov et al., 2012) in R. Depending on the values of X, the response variable Y is assigned to one of 20 clusters, whose cluster means range from 10 to 200 : $cluster.mean \in \{10,\ 20,\ 30,\ 40, ...,\ 190,\ 200\}$. For more information on data generation, please refer to the MixSim package.

of BLRF: s (number of little forests), $ntree$ (number of trees within each little forest), and γ (the parameter that determines the number of observations used to build a little forest). To better understand the interactions between each of these parameters, we incorporate the analysis of γ into those of s and $ntree$, respectively. In this way, we can better understand the effects of s and $ntree$, holding γ constant, while also comparing the effect of γ with all else constant.

5.2.1 Number of Subsamples (s)

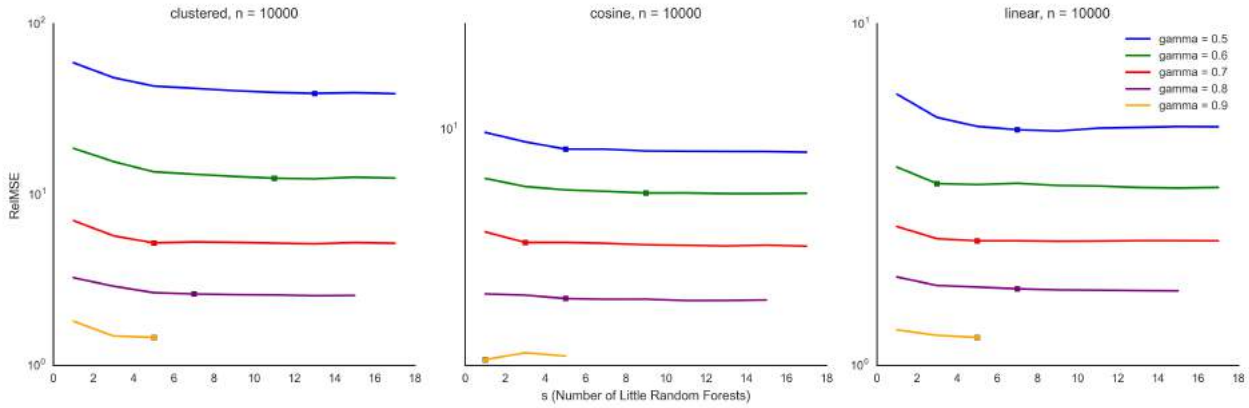

Figure 2: RelMSE (y-axis) Against Number of LRFs (x-axis), s

Figure 2 shows the results for running the BLRF algorithm on the three simulated data sets, holding $ntree = 500$ while varying the values of s and γ. There are three main observations: first, all measures of RelMSE are bigger than one, indicating that the BLRF algorithm overall produces more variable predictions compared to those of the standard Random Forest; second, we observe a decreasing trend in RelMSE as s increases; third, all else constant, RelMSE decreases with an increasing value of γ. All three points are theoretically sound and are to be expected.

For the first finding, since we know from previous chapters that $b << n$, each LRF contains only a subset of all information known by the entire training data set. Therefore, a few LRFs alone should give more variable results than those of the standard Random Forest. However, we should expect asymptotically similar results for the two algorithms, i.e. when γ approaches to 1 (which means each little forest essentially become a standard Random Forest since it has almost all the training data), both models should produce similar results. And indeed, in the figures we see that RelMSE close to 1 when $\gamma = 0.9$ for all data structures.

The second and third findings both have to do with the amount of information known by the entire BLRF. All else constant, if we combine and average the results obtained from more independently constructed LRFs (a larger value for s), we should expect more accurate predictions because the Bag as a whole has more information. Similarly, a larger value for γ means that each LRF has more information; therefore, all else constant, increasing γ will also result in more accurate predictions and a lower RelMSE.

Aside from the above findings, it is also important to discuss the value of s needed for convergence[12] for different values of γ. Kleiner et al. (2014) mentioned "note that fairly modest values

[12]We define "convergence", marked as a square on the plot, as when the drop in RelMSE is less than 1% or when RelMSE goes up.

of s suffice for convergence of BLB, ..., with s at convergence ranging from 1-2 for $b = n^{0.9}$ up to 10-14 for $b = n^{0.5}$". Similar conclusion can be found in our figure above for the clustered data set (the point of "convergence" for each γ value is marked with a square), as s needed for convergence increased from 5 to 13 monotonically as γ goes from 0.5 to 0.9. However, the points of convergence for the Cosine and Linear data sets do not display such a monotonic relationship[13].

In terms of statistical performance, observe from the graphs that small values of γ (0.5-0.6) tend to produce RelMSE too high to be considered useful while larger values of γ (0.7-0.9) produce reasonable (RelMSE close to 1) results given sufficient values of s. More interestingly, there is no point of overlap in between RelMSE curves for different values of γ[14]. Hence, to obtain a relatively good result, we prefer higher values of γ (0.7-0.9) with a sufficient value of s (3 to 10).

5.2.2 Number of Trees ($ntree$)

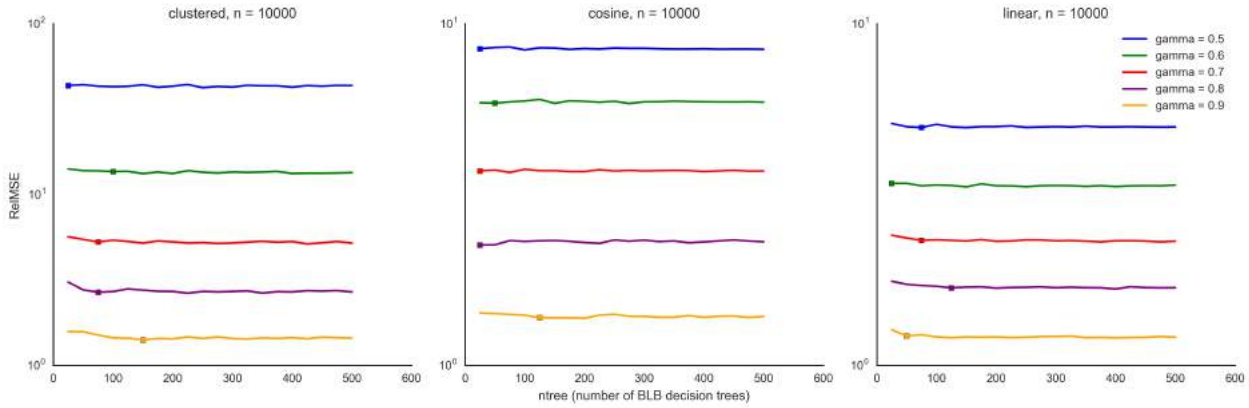

Figure 3: RelMSE (y-axis) against Number of Trees (x-axis) within a LRF, $ntree$

Another important hyperparameter we are interested in is $ntree$, the number of BLB decision trees within each Little Random Forest. Figure 3 shows the results obtained by running BLRF algorithm on three data structures, holding $s = 5$. Similar to the result for s, we observe from the figures that: first, all results obtained are inferior to those of the standard Random Forest; second, larger values of γ are associated with lower RelMSE; third, for each value of γ, increasing $ntree$ tends to lower RelMSE, albeit just slightly.

The reason for the decline in RelMSE resulting from an increasing $ntree$ is much the same as that of the standard Random Forest: a few decision trees cannot extract all available information contained in the given training data set (or a subsample of size b in BLRF's case), while a sufficient number of trees can; also, including more de-correlated trees helps reduce variance and thus stabilizes the entire forest (or one LRF in BLRF's case).

[13]The rationale for such a phenomenon is multitudes. One reason might be that the family of CART algorithm is not particularly suited for modeling data from these two data structures. Thus, we don't need as many LRFs to capture all information that can be captured by a LRF in a subsample.

[14]Although a full explanation of such phenomenon requires more mathematical rigor, intuitively, we can imagine training several weak models using subsets of the training data versus a strong model training using all data. The several weaker learners, each lacking a large portion of information ($b << n$) that the strong model has, will produce inferior results even though we average their results.

However, we do observe some difference in the behavior of BLRF versus that of the standard Random Forest: in the standard Random Forest, usually the sufficient value of $ntree$ for convergence ranges from 300-500, while the range is 100-200 in BLRF (true for all values of γ). This is because the standard RF deals with the entire training data set while each LRF within BLRF deals with a subsample of size $b < n$. This is to say that, the maximum available information that the standard Random Forest can extract is vastly larger that than of a little forest within BLRF. Such finding is of great benefits to us, since if we know that we could use a relatively small value of $ntree$ to obtain a sufficiently good result, we could reduce the amount of computation and thus save execution time.

Therefore, in terms of statistical performance, we prefer a higher value of γ (0.7-0.9) with a sufficient value of $ntree$ (100-200) to achieve results comparable to those of the standard Random Forest.

5.3 Trade-off: Performance versus Time

Along with the discussion of the statistical performance of BLRF, it is equally important to talk about the computational (time) performance of the new model. Again, the time needed to train BLRF directly depends on the choice of three parameters: s, $ntree$ and γ.

5.3.1 Effect of Hyperparameters on Time

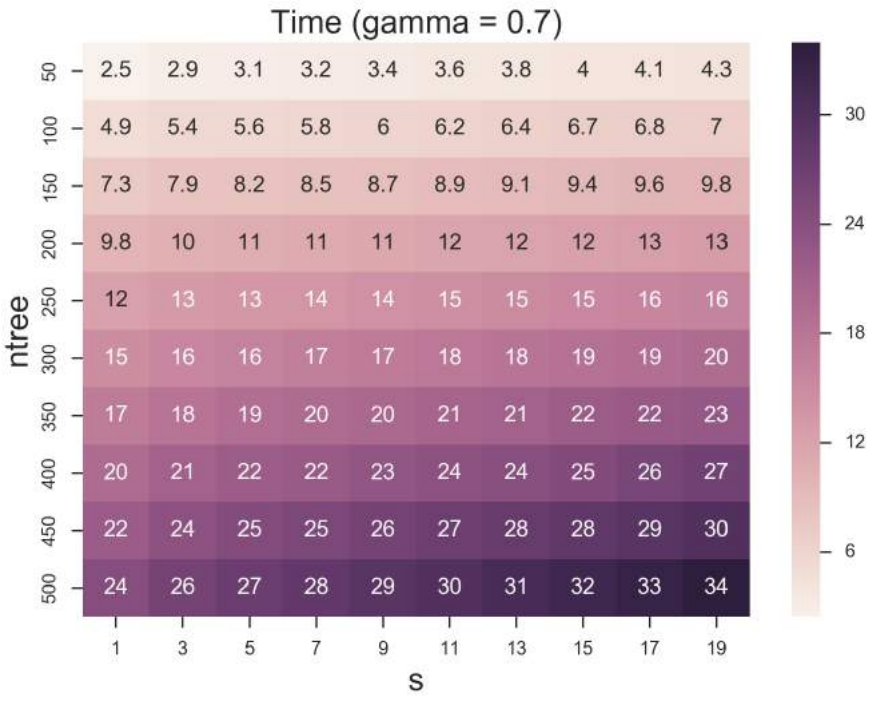

Figure 4: Time (in seconds) Consumed to Train BLRF with Varying Hyperparameters

Figure 4 displays the training times for BLRF with varying values of s and $ntree$ while keeping γ constant at 0.7. It is important to distinguish the effects that these three variables have on the overall time consumption:

s: according to the BLRF algorithm, the s LRFs are all built in parallel, which means that we should not observe substantial increase in time as s increases. In Figure 4, we observe that as s goes from 1 to 19, the training time of BLRF goes up by less than 50% across all values of $ntree$. The reason behind such increase has to do with the extra overhead created by distribution of tasks and aggregation of results, a well-documented trade-off in parallel computing (Barney, 2017).

$ntree$: within each LRF, the BLB decision trees are still built serially. Therefore, time required to build BLRF should scale linearly with $ntree$, which is indeed what we observe.

γ: generally speaking, a larger value of γ leads to a larger b[15], which requires more time as each LRF is larger in size. This time increase should scalre exponentially, as γ appears as the exponent when calculating b (can be seen in Figure 5).

5.3.2 Time Consumption: BLRF vs RF

We would also like to compare the training time of BLRF relative to that of RF.

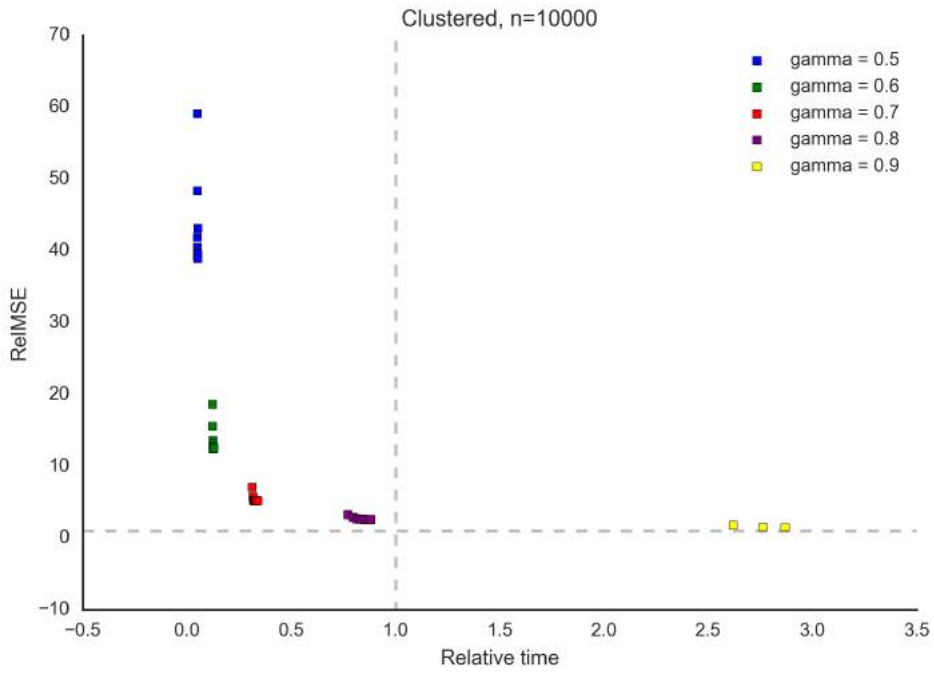

Figure 5: RF versus BLRF: RelMSE and Relative Time

Figure 5 shows the results of a particular simulation using $n = 10000$ observations from the cosine data structure. We trained multiple BLRF models and a RF model using the optimal values of s found in section 5.2.1 for BLRF and $ntree = 500$ for both models. The x and y-axes respectively show the $RelMSE$ and $Relative\ Time$ (time to train a BLRF divided by time to train a RF) of the models. Note that the grey dashed lines represents RelMSE = 1 and Relative Time = 1, or in other words, the baseline performance of the standard RF in terms of accuracy and time. We observe that for $\gamma < 0.9$, BLRF requires significantly less time to train than RF but suffers from higher RelMSE. For $\gamma \geq 0.9$, BLRF took more time to train but the RelMSE is around the same level as RF,

[15] Recall that $b = n^{\gamma}$

suggesting decent asymptotic result. The fact that BLRF with $\gamma \geq 0.9$ requires more time to train presents a problem: the use of BLRF is no longer justified. We believe that this extra computation time is a result of the serial part of the program that cannot be paralleled and the extra overhead created by employing a paralleled structure. It is also possible that the modifications written in C (discussed in section 4.4) were not optimized to their full extent. However, more research is needed on this front.

Based on the above results, we conclude that for practical use, $\gamma = 0.7$ or 0.8 seems to be optimal choices in terms of both time and accuracy: $\gamma = 0.7$ reduces the computation time by more than 60%, while $\gamma = 0.8$ gives better accuracy (around 2 RelMSE).

5.4 Results

To bring the above analyses together, the three hyperparameters, γ, s and $ntree$ all play crucial roles in determining how well the new BLRF model performs in terms of computational time and accuracy. We prefer higher values of γ that ranges from 0.7 to 0.8 to ensure accuracy while picking sufficient s (3-14) and $ntree$ (100-200) values to ensure the convergence (in RelMSE) behavior. Larger values of $\gamma \geq 0.9$ create higher-than-justified computational burden. When the parameters are set as indicated, the BLRF is able to produce results comparable to those of the standard Random Forest (reflected by RelMSE values close to 1) using reduced computational time.

6 Conclusion and Discussion

In this paper, we introduce the new machine learning algorithm - Bag of Little Random Forests (BLRF). We have discussed the mechanisms of BLRF as well as its two fundamental building blocks: Random Forests (RF) and Bag of Little Bootstraps (BLB). We have also show through simulations that BLRF is able to reduce the model training time while maintaining levels of accuracy comparable to those of RF.

However, there are several necessary modifications to our current version of BLRF. First, in our implementation of BLRF, the context is strictly limited to a regression setting. To make the model fully functional, it is necessary to also implement the classification model of BLRF. Second, we have not yet dealt with variable importance or OOB error estimation, both important components of the RF algorithm. Finally, we are not entirely clear on the reasons behind the larger time consumption required by BLRF when $\gamma \geq 0.9$.

Future work indicates completing the BLRF package by adding the classification part of the algorithm, as well as calculating variable importance and OOB error rates. To better address the computational time issues, we will start building the BLRF package on top of the $ranger$ (Wright and Ziegler, 2015) package, a C++ implementation of Random Forests that is easier to work with than the original $randomForest$ package[16].

[16]The reason that we did not implement classification BLRF is that the original RF R package was written in C (for regression) and Fortran (for classification). The learning curves for both were quite high. However, both regression and classification of RF in the $ranger$ implementation are written in C++ and are better-documented.

References

Barney, B. (2017). Introduction to parallel computing. *Lawrence Livermore National Laboratory*.

Breiman, L. (1 October, 2001). Random forests. *Machine learning, Springer Netherlands*, 45:5–32.

Caruana, R., Karampatziakis, N., and Yessenalina, A. (2008). An empirical evaluation of supervised learning in high dimensions. In Cohen, W. W., Mccallum, A., and Roweis, S. T., editors, *Proceedings of the 25th International Conference on Machine Learning (ICML-08)*, pages 96–103.

Efron, B. and Tibshirani, R. (May 15, 1994). *An Introduction to the Bootstrap*. CRC Press, illustrated, reprint edition.

Gareth James, Daniela Witten, T. H. and Tibshirani, R. (2013). *An Introduction to Statistical Learning with Applications in R*. 1431-875X. Springer-Verlag New York, 11 edition.

Kleiner, A., Talwalkar, A., Sarkar, P., and Jordan, M. I. (September 2014). A scalable bootstrap for massive data. *Journal of the Royal Statistical Society*, Volume 76, Issue 4:Pages 795 to 816.

Leo Breiman, Jerome H Friedman, R. A. O. C. J. S. (May 1999). *Classification and Regression Trees*. CRC Press, New York.

Liaw, A. and Wiener, M. (2002). Classification and regression by randomforest. *R News*, 2(3):18–22.

Melnykov, V., Chen, W.-C., and Maitra, R. (2012). MixSim: An R package for simulating data to study performance of clustering algorithms. *Journal of Statistical Software*, 51:12.

Mjolsness, E. and DeCost, D. (2001). Machine learning for science: State of the art and future prospects. *Science, New Series*, Vol. 293(No. 5537):No. 5537.

Wright, M. N. and Ziegler, A. (2015). ranger: A fast implementation of random forests for high dimensional data in c++ and r. *arXiv:1508.04409v1 [stat.ML]*.

Yenny Zhang, D. J. H. (2017). Integrating random forests into the bag of little bootstraps. Submitted to [redacted] in Partial Fulfillment of the Degree of Bachelor of Arts.