Pomona College

Bag of Little Random Forests (BLRF)

Author:
Zihao Xu

Advisor:
Dr. Johanna Hardin

Submitted to Pomona College in Partial Fulfillment
of the Degree of Bachelor of Arts

September 15, 2017

Abstract

Random Forests are a successful ensemble method that utilizes a number of decision trees to make predictions robust in both regression and classification settings. However, the process of bootstrap aggregation, the mechanism underlying the random forest algorithm, requires each decision tree to physically store and perform computations on data sets of the same size as the input training set, a situation that is oftentimes impractical given the humongous sizes of data sets today. To address this problem, we introduce the Bag of Little Random Forests (BLRF), a new algorithm that adapts the Bags of Little Bootstraps (Kleiner el al.), aiming to achieve a better computational profile while producing predictions with comparable accuracy as those of random forest.

Contents

Chapter 1

Introduction

With the advent of the era of big data, machine learning techniques like K-nearest neighbors, Support Vector Machine and Random Forests have become prevalent in solving problems across areas ranging from astrophysics to pharmaceutics. Equipped with powerful techniques, data scientists and business analysts have managed to investigate the secrets of financial markets and consumer behavior.

Random forests [] are among one of the top performing machine learning methods that is robust in numerous regression and classification settings (Caruana, last part of Section 4, http://lowrank.net/nikos/pubs/empirical.pdf). Essential to the random forests algorithm is the idea of bootstrap aggregation, also known as "bagging"[**?**], which means the forests consist of an number of decision trees at training time and output the mode or the average results of all the trees. Bootstrap aggregation is powerful in that it mitigates the well-known variation and over-fitting problems of decision trees, yielding high accuracy.

Albeit their effectiveness, many machine learning algorithms oftentimes 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 is often massive. The challenge to address such a problem makes an alternative resampling method stand out, Bag of Little bootstraps().

Bag of Little bootstraps, also known as BLB, is a resampling method devised by Kleiner et al. [].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, pages 12]. 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 the modern 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 random forests 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 performance in Chapter 5, followed by a discussion on its computational profile in Chapter 6; finally, we will conclude with some further discussion on possible modifications and future directions in Chapter 7.

Chapter 2

Random Forests

As we saw in Chapter 1, the random forests algorithm is a powerful machine learning algorithm that can be applied to both classification and regression settings. More importantly, random forests are 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 further transformed into a random Forest. 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 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 finite number of *regions*, or *nodes*, that represent the most homogeneous space with respect to the response variable for the data. As a result, the model can be conveniently represented graphically as a decision tree, as we will see below. Since this paper is mostly concerned with regression, the following sections are dedicated to the fundamentals of CART regression trees. Classification trees will be briefly discussed at the end of this section.

2.1.1 Regression tree

In simple terms, the regression tree is built in the following way:

1. First we divide 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 $R_1, R_2, ..., R_J$, to minimize the Sum of Squared Errors (SSE) for the tree, given by:

$$SSE_{tree} = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \bar{y}_{R_j})^2 \tag{2.1}$$

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 value means that the responses in a region are spread farther apart from each other than if those with a lower SSE. Thus, the best tree minimizes the SSE of the model.

2. To predict the value of a new test observation, follow down the tree model constructed above 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, or nodes, are constructed.

2.1.2 Recursive Binary Splitting

The most important procedure, and the part that is most relevant to our construction of Bag of Little random forests (as we shall see in Chapter 4), is called recursive binary splits.

This approach is *top-down* and *greedy*: "$top - down$" because we begins the splits at top of the tree (where all observations belong to a single region) and then successively split each cluster of points into two branches further down the tree; "*greedy*" because at each split, the model look through all predictors and all of its possible splits to determine the *best* one so that the sum of squares for the response variable is minimized.

More specifically, to perform recursive binary splitting on a given node, t, we go through each one of $X_1, X_2, ...X_p$ and all possible cutpoints, s (gaps in between each value of the response variable), that splits the observations within that node into two distinct regions : $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$. 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:

$$\Delta SSE = SSE_t - (SSE_{t_R} + SSE_{t_L}) \tag{2.2}$$

In a regression setting, the CART algorithm will iterate such binary splitting procedure recursively 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 population 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):

$$\begin{aligned}
SSE_t &= \sum_{i \in R_t} (y_i - \bar{y_t})^2 \\
&= \sum_{i=0}^{n_t} (y_i^2 - 2 * y_i * \bar{y_t} + \bar{y_t}^2) \\
&= \sum_{i \in R_t} y_i^2 - 2 * \sum_{i \in R_t} y_i * \frac{SUM_t}{n_t} + \sum_{i \in R_t} \left(\frac{SUM_t}{n_t} \right)^2 \\
&= \sum_{i \in R_t} y_i^2 - 2 * \frac{SUM_t^2}{n_t} + n_t * \left(\frac{SUM_t}{n_t} \right)^2 \\
&= \sum_{i \in R_t} y_i^2 - 2 * \frac{SUM_t^2}{n_t} + \frac{SUM_t^2}{n_t} \\
&= \sum_{i \in R_t} y_i^2 - \frac{SUM_t^2}{n_t}
\end{aligned} \tag{2.3}$$

With this, Equation 2.2 can be rewritten as:

$$\begin{aligned}
\Delta SSE &= SSE_t - (SSE_{t_R} + SSE_{t_L}) \\
&= \sum_{i \in R_t} y_i^2 - \frac{SUM_t^2}{n_t} - \left(\sum_{i \in R_{t_L}} y_i^2 - \frac{SUM_{t_L}^2}{n_{t_L}} + \sum_{i \in R_{t_R}} y_i^2 - \frac{SUM_{t_R}^2}{n_{t_R}} \right) \\
&= \frac{SUM_{t_L}^2}{n_{t_L}} + \frac{SUM_{t_R}^2}{n_{t_R}} - \frac{SUM_t^2}{n_t}
\end{aligned} \tag{2.4}$$

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

Therefore, to find ΔSSE, we simply need to calculate the difference in $\frac{SUM}{N}$ 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 (taking the mean 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 set error at the cost of an increased test set error.

Therefore, the algorithm of random forests is proposed to address such problems by introducing two "sources of randomness": bootstrap aggregation and de-correlation.

2.2.1 Bootstrap Aggregation

Bootstrap aggregation, also known as bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy. In the random forest setting, the idea basic 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 bootstrapping, a resampling method that measures the accuracy (e.g. defined in terms of bias, variance, confidence intervals, or prediction error) to sample estimates. Here, we repeatedly sample with replacement from the given training observations to build a number of decision trees whose size are the *same* as the given training set. The combined effect is that the variability is drastically reduced since the average result for all trees is used as the final result.

2.2.2 Random Forests

On top of bootstrap aggregation, there is one more step towards random forests: de-correlation.

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 for the construction of each tree. This procedure effectively de-correlates the tree because if all the explanatory variables are available for all decision trees, it is highly likely that there are several dominating variables that most of the trees decide to split on and thus become similar. By only give each tree m out of p (often $m = \sqrt{p}$) predictors, such dominance effect would no longer exist and the decision trees de-correlated. Another benefit of such de-correlation is that variance of decision trees is reduced since averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities [text book].

2.2.3 Limitations of Random Forests

Even though random forests effectively mitigate the limitations of CART, they are not without their own drawbacks, notably in memory consumption and computation.

First, 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 since data sets can be too massive to be stored or manipulated in a efficient manner, and random forests require repeated storage of such data sets in memory for building hundreds of tree models.

In addition, also due to the large size of the resamples (for bootstrap aggregation), 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 oftentimes 500 or more times.

Therefore, we are motivated to develop a modification to random forests that drastically reduces the computational burdens 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.

Chapter 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 [**?**]. However, because each resample is of the same size as the original data set, with roughly 63 percent of the observations present, performing computation for even a single point estimate on large data sets presents computational challenges. Here, one might naturally turn to the modern parallel and distributed computing structure, 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 can be overly high, as is the cost of operating on even a single resample using an independent set of computing resources” [Kl].

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 into a resampling method well-suited for computations on large data sets, while maintaining the favorable statistical properties of the bootstrap [KL]. In this chapter, we will briefly talk about BLB's notation, procedures and advantages. For a more rigorious discussion on BLB's statistical properties (consistency and higher-order correctness), refer to .

3.1 Setting and Notation

Assume we have a set of training observations, $X_1, X_2, ..., X_n$, drawn from an unknown population P. We denote its empirical distribution as $\mathbb{P}_n$. 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 $\hat{\theta}(\mathbb{P}_n)$ to indicate that we used data from distribution $\mathbb{P}_n$ to compute the estimate, $\hat{\theta}_n$.

Recalling from Section 2.2.1, the bootstrap aims to estimate the true sampling distribution of $\hat{\theta}_n$ (call it $Q_n(P)$), from which we can compute a quality assessment, $\xi(Q_n(P), P)$, which could be a confidence interval, standard error, or bias. For simplicity, we use $\xi(Q_n(P))$ to replace $\xi(Q_n(P), P)$. The bootstrap computes the approximation $\xi(Q_n(\mathbb{P}_n)) \approx \xi(Q_n(P))$ by approximating $\xi(Q_n(\mathbb{P}_n))$. The bootstrap takes repeated resamples of size n from $\mathbb{P}_n$ and forms the empirical distribution, denoted by $\mathbb{Q}_n^*$; then it computes $\hat{\theta}_n$ on each resample, and finally completes the approximation $\xi(\mathbb{Q}_n^*) \approx \xi(Q_n(\mathbb{P}_n)) \approx \xi(Q_n(P))$.

3.2 Bags of Little bootstraps (BLB)

3.2.1 Mechanism

In the simplest terms, BLB functions by combining the results of bootstrapping multiple small subsets of the larger sample.

Before we proceed, it is useful to define the parameters that we will be using 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 3.1: Parameters of interest

First, we pick the value of γ 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 [1]. In this way, all the observations within each subsample are distinct, whereas each observations may be present in

[1]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.

multiple subsamples. 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_j is $\mathbb{P}_{n,b}^{(j)}$. After this, we bootstrap on each subsample **up to size** n, to obtain the quality assessment $\xi(Q_n(\mathbb{P}_{n,b}^{(j)}))$; finally, the BLB's estimate of $\xi(Q_n(P))$ is given by:

$$\xi(Q_n(P)) = \frac{1}{s}\sum_{j=1}^{s}\xi(Q_n(\mathbb{P}_{n,b}^{(j)})) \qquad (3.1)$$

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

Recall from the beginning of the chapter that one of the major challenges associated with 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 needs the size of resamples to be n, which can be potentially troublesome. Fortunately, since the subsamples in the BLB method are all of size $b < n$, the multinomial distribution can be employed to mitigate the sample size discrepancies.

The multinomial method makes use of the multinomial distribution, a generalized version of binomial distribution that is used to model situations like the probability of counts for rolling a k-sided die n times. Given a total 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 random variable coefficients $\mathbf{M} = \{M_1, M_2, ..., M_b\}$. In the BLB's setting, n represents the size of the given training data set and $\mathbf{p} = \{1/b, 1/b, ..., 1/b\}$ since each observation should have equal probability of being chosen (akin to the bootstrap). Therefore, each M_i in the vector $\mathbf{M}$ represents the number of time that observation X_i is present in the resample of the subset.

To better illustrate the multinomial method, figure 3.1 shows an example of a resample from subset of size $b = 5$ up to size $n = 100$. In this case, the Multinomial Method produces a vector of multinomial coefficients $\mathbf{M} = \{21, 15, 22, 23, 19\}$, each representing the number of times an observation is present in this resample.

cite Yenny

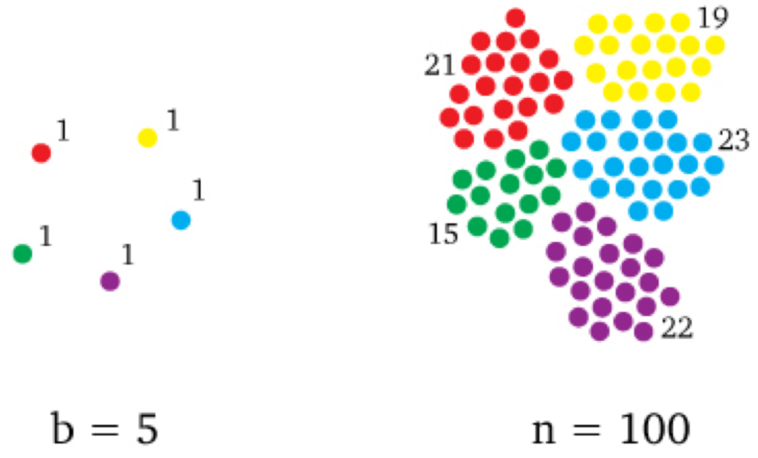

Figure 3.1: A Resample of one subsample of the Training Observations

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 of the subsample is of size b, which in practice is much smaller [2] than n , the computation associated with BLB scales in $\mathbf{O}(b)$, as opposed to $\mathbf{O}(n)$ [Kleiner].

In terms of memory usage, since each subsample is of size $b << n$, the memory required by BLB is much smaller than that required by bootstrapping on the original data directly. Such reduction in memory requirement results from the fact that each resample (of size b) drawn from the subsamples are much smaller than a resample (of size n) from the standard bootstrap. However, it is important to note that, each resample in BLB requires not only a copy of subsample (of size b) of observations but also a vector (size b) of multinomial coefficients. Such additional requirement to store a vector of multinomial coefficients may seem potentially cumbersome, but as Kleiner

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

et al. mentioned, for example, when $\gamma = 0.9$, we need only $s = 2$ subsamples to obtain a result comparable to that of bootstrap. In such a situation, the combined memory usage of BLB is much smaller than that of the bootstrap [3]. Another huge benefit of BLB is that, if the bootstrap has to deal with data sets of too large sizes (which means each resamples will also be large), sometimes it is even impossible to store such data sets nore perform any meaning computation; in contrast, the BLB method makes such computation possible by reducing the memory needed (that for b observations within the subsample and a vector of b multinomial coefficients) for each resample from the subsample.

Third, with such a setup, BLB is more suited than the bootstrap to utilize paralleled and distributed computing architecture. There are several options to choose from: passing subsamples to multiple cores, processing subsamples serially but pass each resample of the subsamples to multiple cores, and passing all resamples of subsamples to multiple cores at the same time (requires a large numb re of cores). One could argue that the bootstrap can also utilize such paralleled structure to perform its computation. But again, the massive size of the training observations might be too big for any single core to process. However, the BLB method does not have such a hurdle and can thus reduce computation time drastically.

[3]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. And the memory needed for convergence for smaller values of γ is even smaller

Chapter 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 random forests with one major difference: instead of using bootstrap aggregation (or bagging), we will use "BLB aggregation" to build multiple forests each built from many 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, for each subsample, build a "Little Random Forest" using the observations specific to this forest and the multinomial method.

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, or RMSE, which will be defined and further discussed in Chapter 5) comparable to that of the standard random forests algorithm.

In this chapter, we will go from the macro to the micro: first we will talk about the construction of the Bag of Little Random Forests and the relationship between the Little Forests; 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 comparison of BLRF to random forests and random forests to CART.

4.1 BLB aggregation

As its name suggests, “Bag of Little Random Forests” consist of a number of “Little Forests” and the final result is the aggregation of all the forests. More specifically, in a regression setting, each “Little Forest” uses a subsample of training observations and the multinomial method to produce a prediction for a test observation, while the aggregated result given by the BLRF algorithm is the average of results given by all the “Little forests”. 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:

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

Table 4.1: Parameters in BLRF

Note that all of the symbols are kept the same except r, which is now denoted *ntree*. This is because if we were to replace bootstrap aggregation with BLB aggregation, since each subsample corresponds to a single Little random Forest, the corresponding parameter of r, representing number of re-samples of each subsample, should be *ntree*, representing number of decision trees within each Little random Forest.

4.1.1 Construction of the Bag

First, it is important to understand how to construct the Bag of Little forests and what is the relationship in between the Little forests.

Given a set of training observations, $X_1, X_2, ..., X_n$, of size n, we repeatedly subsample, or sample without replacement, s *distinct* subsets of size b from the training observations, where $b = n^\gamma$ and $\gamma \in [0, 1]$ is a parameter defined by the user [1]. 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 "Little random forests" are not independent[2].

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"

Much like the standard random forests algorithm, the building block of a "Little Random Forest" 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 Little Random Forest, 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 Little random Forest to a number of (usually 500) decision trees. With that, we can delve into how a "BLB" decision tree is constructed within a Little random Forest.

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

[2]Like BLB (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 $b = n\gamma = 500^{0.9} = 268$, there can be one such subsample, but we need two or more subsamples to obtain a decent result. The example of $n = 500$ may not, however, be relevant because BLRF is most suited for much larger data sets.

4.1.3 A “BLB” decision tree

As discussed in Section 2.1.2, the crux of the CART algorithm is the 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 2.4.

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 Little Random Forest 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 a BLB decision tree is: the b training observations assigned to the Little Forest and a multinomial coefficients vector. 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 2.4 (reproduced below), though with the addition of the multinomial coefficients, the calculations of the node sum, SUM, and the node population, n, are different:

$$\begin{aligned} \Delta SSE &= SSE_t - (SSE_{t_R} + SSE_{t_L}) \\ &= \frac{SUM_{t_L}^2}{n_{t_L}} + \frac{SUM_{t_R}^2}{n_{t_R}} - \frac{SUM_t^2}{n_t} \end{aligned} \tag{4.1}$$

The new SUM and n are calculated as follows:

$$SUM = \sum_{i=1}^{b} y_i * M_i \tag{4.2}$$

$$n = \sum_{i=1}^{b} M_i \tag{4.3}$$

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 4.3 - that is, stop the split if the sum of the multinomial coefficents associated with the observations within a given node is less than 5.

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 reduce the amount of computation, mainly due to the fact that each Little random Forest 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 4.1 for $0.632n$ times [3]. However, inside a Little random Forest, 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 4.1 is strictly less than $b - 1$, which in practice is much smaller than $0.632n$ [4]. Here, one may validly argue that, Bag of Little random forests needs to build a number of Little random forests, 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 Little random Forest deals with, the amount of memory required for each Little Forest 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

[3]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 4.1 is equal to $0.632n$.

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

Forest to work with. Facing a data set too large to be stored in memory at once, BLRF is able to handle such a data set since each Little random Forest only requires a *subset* of the entire data set, whereas the original random Forest cannot perform any computation or give predictions.

Another major benefit that BLRF has over random Forest is that, the computation of BLRF is more suited for the modern distributed and paralleled structure. As discussed in Section 3.3, we can pass each subsample, or a Little random Forest in our case, to an individual core and aggregate the results after the computation is done. If we have the computational resources of several multi-core computers, we can distributed each Little random Forest to each computer wherein the resamples, or BLB decision trees, can be built using the computing power of a number of cores. The distributed and paralleled structure also alleviates the computational burden brought by the need to construct a number of Little random Forest by utilizing the power of several computing cores and even several computers.

On top of these three advantages, BLRF also provides a new measure of variability for the prediction that random Forest does not have. Previously, with only one Forest, one cannot say much about the variability or confidence of a certain prediction. However, for BLRF, we have s predictions given by s Little random forests, whose average is the final prediction. We can further use such a fact and approximate the variability of the final prediction by calculating the variability of s results.

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

4.3 Higher Order Comparison

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. On top of this, BLRF uses BLB aggregation and the predictability of several Little random forests 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.

Chapter 5

Statistical 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 is necessary for the control of sample size and knowledge of the underlying relationship between the response and explanatory variables. In addition, to construct the Relative MSE (RMSE) 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 better 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 robust algorithm.

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 the results obtained by the standard random forest algorithm. Since we are working exclusively within a regression setting, the normalized results are measured in $RMSE$, here :

$$RMSE = \frac{MSE_{BLRF}}{MSE_{RF}} \tag{5.1}$$

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

All tests below 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:

$$\begin{aligned}
&linear\text{: } Y = 5X_1 + 10X_2 + 15X_3 + 20X4 + 25X_5 + \varepsilon \\
&consine: Y = 50 \times cos(\pi \times (X_1 + X_2)) + \varepsilon \\
&clustered\ ^2\text{: } Y = cluster.mean + \varepsilon
\end{aligned}$$

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

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 little random forests, s, since each little forest should be built independently using a single core. In the simulation. In our simulation, s is at most 18.

5.2 Effect of Hyperparameters

With an overall understanding of the simulation setup, we are ready to discuss the simulation results. There are mainly three hyperparameters that we are interested in: 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

[1]We define "optimal" as the average MSE of the random forest algorithm with 10 iterations with $ntree = 500$.

[2]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 : $cluster.mean \in \{10,\ 20,\ 30,\ 40, ...,\ 190,\ 200\}$.

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)

This graph needs to be modified.

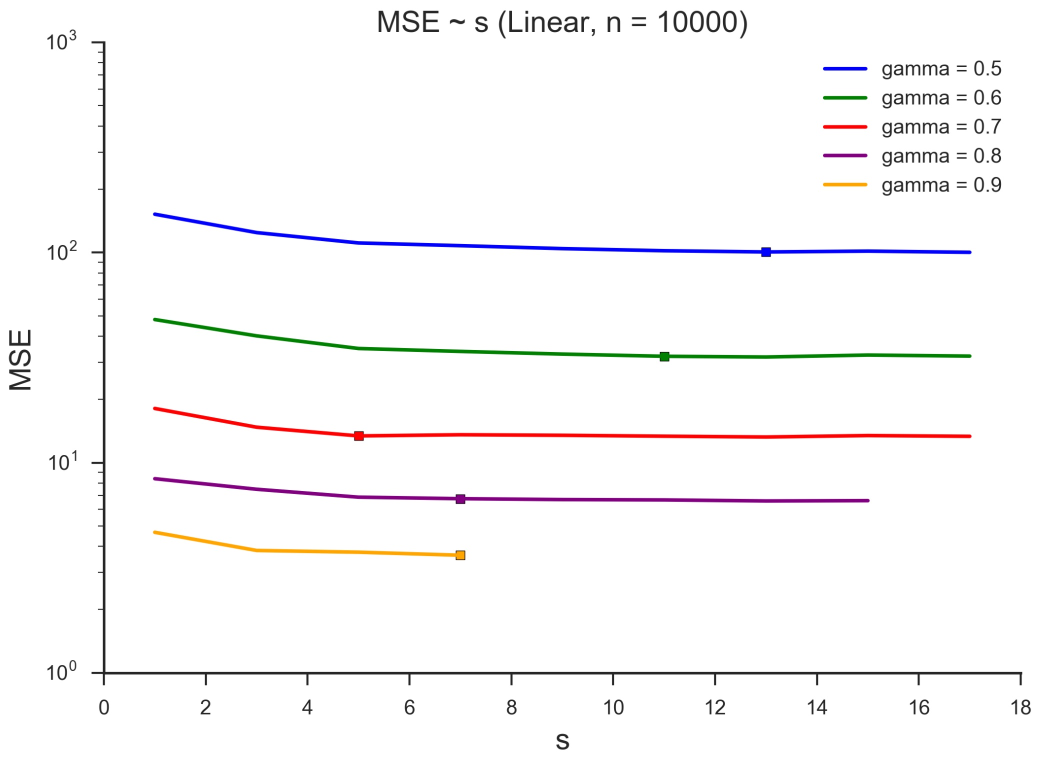

Figure 5.1: A Resample of one subsample of the Training Observations

Figure 5.1 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[3] from this plot: firstly, all measures of RMSE are bigger than one, indicating that the BLRF algorithm overall

[3]What we observe from the graph.

produce inferior results compared to those of the standard random forests; secondly, we observe a decreasing trend in RMSE as s increases; thirdly, all else constant, RMSE decreases with an increasing value of γ. All three points are theoretically sound and are to be expected:

For the first point, since we know from previous chapters that $b << n$, each little forests should only contain a subset of all information known by the entire training data set. Therefore, a few little forests alone should give inferior results than those of the standard random forest. However, it is entirely possible for BLRF to give better results than the standard random forests, especially when γ is close to 1 (which means each little forest know nearly all information available), but we have not observed such cases in our simulation.

The second and third point 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 little forests (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 little forest has more information; therefore, all else constant, increasing γ will also result in more accurate predictions and lower RMSE.

Aside from the above findings, it is also important to discuss the value of s needed for convergence [4] for different values of γ. In Kleiner et al.'s paper, they mentioned that, "note that fairly modest values 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 also be found in our figure above (the point of "convergence" for each γ value is marked with a square), as s needed for convergence ranges from 10-14 to 3-5 as γ goes from 0.5 to 0.9.

In terms of statistical performance, oberserve from the graph that small values of γ (0.5-0.6) tend to produce RMSE too high to be considered useful while larger values of γ (0.7-0.9) produce reasonable results given sufficient values of s. 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).

[4] We define "convergence" as when the drop in RMSE is less than 1% or when RMSE goes up.

5.2.2 Number of Trees (*ntree*)

This graph needs to be changed.

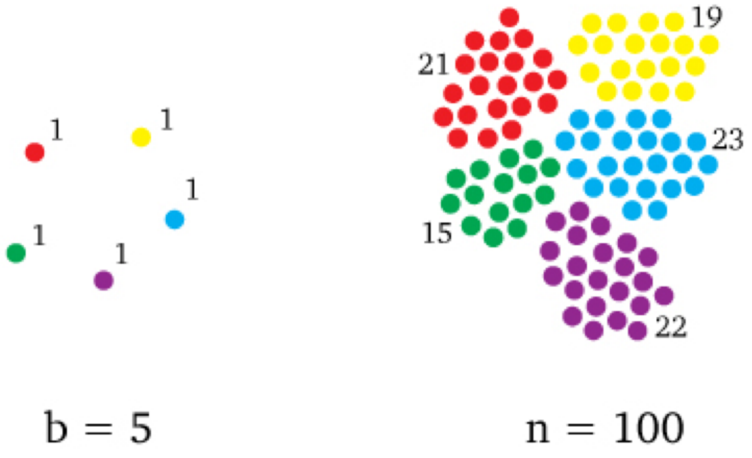

Figure 5.2: A Resample of one subsample of the Training Observations

Another important hyperparameter we are interested in is *ntree*, the number of BLB decision trees within each little random forest. Figure 5.2 show the results obtained by running BLRF algorithm on three data structures, holding $s = 5$. Similar to the result for s, we observe from the graph that: firstly, all results are inferior to those of the standard random forst; secondly, larger values of γ are associated with lower RMSE; thirdly, for each value of γ, increasing *ntree* tends to lower RMSE.

The reason for the decline in RMSE resulting from an increasing *ntree* is much the same as that of the standard random forest: few decision trees cannot extract all avaiable information contained in the given training data set (or a subsample of size b in BLRF's case), while a sufficiant number of tree can; also, including more de-correlated trees helps reduce variance and thus stablizes the entire forest (or a little forest in BLRF's case).

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 deal with the entire training data set while each little random forest 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 sufficiant value of *ntree* (100-200) to achieve results comparable to those of the standard random forest.

5.3 Conclusion

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. We prefer higher values of γ that ranges from 0.7 to 0.9 while picking sufficient s (3-7) and *ntree* (100-200) values. Larger values of s and *ntree* will not give more accurate predictions but will increase computational burden. When the above percedure is followed, the BLRF is able to produce results comparable to that of the standard random forest (reflected by RMSE values close to 1).

However, we are not only interested in the statistical performance of BLRF alone; we have to take into account the amoumt of time needed to obtain the results using the BLRF algorithm. As we will see in the next chapter, there exists a tradeoff between BLRF's performance in terms of RMSE and its computational time. We will try to find a balance between the two and ultimately determine a good choice for values of the three hyperparameters.

Chapter 6

Computational Scalability

6.1 Simulation Setup

6.2 Trade-off: Performance versus Time

6.3 BLRF Versus Random Forest

6.4 Conclusion

Chapter 7

Further Discussion

7.1 Necessary Modifications

7.2 Further Explorations

7.2.1 Out Of Bag (OOB): Model Assessment

[Breiman, 2001] [Caruana et al., 2008] [Ariel Kleiner, 2014] [Bradley Efron, 1994] [Yenny Zhang, 2017] [Wang et al., 2014]

7.2.2 Variable Importance

7.2.3 Data Structures and Performance

Chapter 8

Conclusion

8.1 Conclusion

8.2 Acknowledgement

[?]

Bibliography

[Ariel Kleiner, 2014] Ariel Kleiner, Ameet Talwalkar, P. S. . I. J. (September 2014). A scalable bootstrap for massive data. *Journal of the Royal Statistical Society*, Volume 76, Issue 4:Pages 795 to 816.

[Bradley Efron, 1994] Bradley Efron, R. T. (May 15, 1994). *An Introduction to the Bootstrap*. CRC Press, illustrated, reprint edition.

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

[Caruana et al., 2008] 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.

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

[Wang et al., 2014] Wang, H., Zhuang, F., and He, Q. (2014). Scalable bootstrap clustering for massive data. *SNPD*.

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