A Survey on Tree-Based Machine Learning Methods: The Path to XGBoost and Generalized Random Forests

Andong Yan
University of California, Riverside
Riverside, CA 92521
ayan013@ucr.edu

May 30, 2020

1 Introduction

XGBoost is a famous algorithm that is widely used for machine learning tasks to achieve state-of-the-art results in regression and classification problems. It is an implementation of gradient boosted decision trees and is specifically designed for speed and performance, which help it dominate in many machine learning competitions and become one of the most popular algorithms in the machine learning community.

Susan et al (2018) propose a generalized random forests method for non-parametric statistical estimation based on random forests (Breiman, 2001). Their new method aims to fit any quantity of interest identified as the solution to a set of local moment equations by considering a weighted set of nearby training examples. The innovation of this paper locates at the fact that, instead of using classical kernel weighting functions that are prone to a strong curse of dimensionality, they use an adaptive weighting function constructed by their proposed random forests algorithm that is designed to capture the heterogeneity in the specified quantity of interest. They also provide a computationally efficient algorithm for growing the generalized random forests that is asymptotically converging to the theoretical method. Moreover, they develop a large large sample theory for the consistency and asymptotically Gaussian property and provide an estimator for their

asymptotic variance that enables valid confidence intervals (which involves intense technical details in deriving the inference and will not be covered by this note).

This paper will summarize the essence of these two latest tree-based machine learning methods: *XGBoost* and *Generalized Random Forests*, and other related foundational methods such as Adaboost, random forest and so on.

To understand the algorithm of the two algorithms mentioned above, we need to first review the ideas of their major components, as they are also important algorithms in the field of machine learning and statistics. This literature review will first illustrate the basic idea of a decision tree in machine learning tasks and introduce traditional Classification And Regression Trees(CART) methods as the basic applications of the tree method in the learning tasks. Then bootstrap aggregating(as known as bagging), a simple idea of an ensemble of learning procedures to improve the accuracy of the learning algorithm, and boosting, which differs with bagging in their ways of choosing new training samples, will be discussed. I will also illustrate Adaboost, a basic application of boosting, and Gradient boosting, which optimizes the learning procedure based on an additive model with a suitable loss function. In contrast to boosting, forest methods use a different direction of tree aggregation. I will summarize the random forest method, which is one of the most successful forest implementations. Finally, XGBoost and Generalized Random Forests combine many features of previous boosting and forest methods and optimizes their systems to achieve a higher performance in running time and accuracy.

In section 2, I will summarize the concept of decision tree and CART methods. In section 3, I will explain the concept of bagging and boosting as well as the convergence and consistency result on boosting. In section 4, I will provide an illustration on the algorithm of Adaboost and gradient boosting and their link to the additive model from a statistical perspective. In section 5, I will explain the random forest method, the convergence and consistency result of its generalized method and its difference from boosting methods. In section 6, XGBoost will be formally introduced as well as its major components and innovations. In section 7, Generalized Random Forests will be briefly illustrated with several simulation and empirical applications.

2 Decision Tree and CART

A decision tree is an algorithm to classify a sample into subsets and predict the value of a target variable based on the predicted value in each subset. In the decision tree, each internal node represents a class of data that is classified by several input features(variables) and before reaching the terminal node(leaf), the node needs to split into subsets according to certain splitting rules. At the terminal nodes(leaves), the values of the targeted variable will be generated for the partitioned data in each leaf, and the final prediction of the whole sample will be aggregated over the leaves.

Figure 1 displays a simple example of a decision tree to estimate the survival rate of the passengers in the sinking of Titanic, representing the original idea of the first regression tree algorithm, proposed by Morgan and Sonquist (1963) in their Automatic Interaction Detection (AID) model. By recursively splitting data into two nodes, with splits of form "$x_i \leq c$", where x_i is one dimension of the features and c is some threshold value, the regression tree fits a piecewise-constant model. At each node t, define node impurity function $\phi(t) = \sum_{i \in t}(y_i - \bar{y})^2$, and the tree chooses split to minimize the sum of impurities of two nodes by greedily searching for all candidates at each split. The tree will stop splitting if the reduction in node impurity is lower than a threshold, and the predicted value at each leaf is the average value of all samples in the subset that the tested sample falls in. The important fact is that any test sample could only fall in exactly one leaf and the average of the training samples in that leaf could provide a close estimate to the test sample.

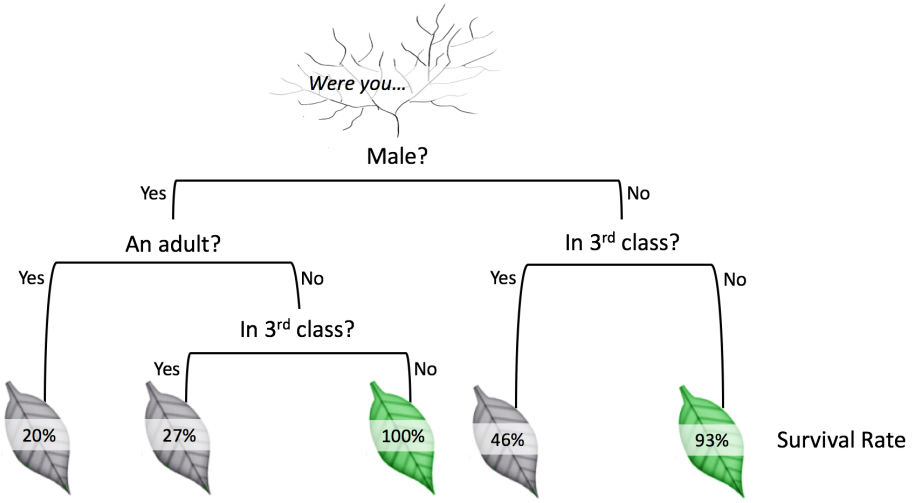

Figure 1: Decision Tree to Predict whether you'd survive the sinking of Titanic, *Source*: https://algobeans.com/2016/07/27/decision-trees-tutorial/

Another classical decision tree method that could be used for both classification and regression tasks is the Classification And Regression Tree (CART) introduced by Breiman et al. (1984). CART carries the idea of recursive splitting the sample data to minimizing the selected objective function with splitting and stopping rules from AID. One problem of AID is that when to stop the splitting or the optimal depth of the tree: too many splits performed on the training sample could result in small sizes of samples in the leaves, which increases the variance in the sample means due to the nature of a small sample, or "overfitting" in machine learning terminology. The key innovation in CART is how to make more accurate splits and build the tree with less complexity.

Breiman et al. (1984) propose a pruning technique to penalize the complexity of the tree structure and a linear split rule to separate the sample space more flexibly. While the stopping rule aims to control the depth of the tree when choosing the split, pruning could help eliminate inefficient splits after the training finished. One pruning technique is the cost-complexity pruning, which defines the cost-complexity measure $R_\alpha(T)$ for the tree T grown from the original algorithm:

$$R_\alpha(T) = R(T) + \alpha|T|$$

where $R(T)$ is the object function to minimize or the cost function, $|T|$ is the number of leaf nodes in the tree and $\alpha \geq 0$ is the complexity parameter. When $\alpha = 0$, as no penalty on the tree complexity, the original tree will be chosen; when $\alpha > 0$ increases, the pruned tree $T(\alpha)$ that minimizes $R_\alpha(T)$ will be chosen from a series of subtrees of T that recursively prunes a pair of leaves from the same parent node. Finally, the choice of the complexity parameter α could be determined by cross-validation.

The linear split rule replaces the splits "$x_i \leq c$" by linear splits "$\sum_i a_i x_i \leq c$", essentially allowing any hyperplane to separate the sample place into two regions instead of only the hyperplanes orthogonal to the axes. This extension makes each split more efficient as it provides more flexibility when choosing the split to minimize the overall cost function.

Despite the effort of CART to make the tree method more efficient and less complex, it still suffers from selection biases and lack of robustness. First, the greedy algorithm to pick the split could only guarantee a locally optimal decision at each node but not the global optimum. Second, it is not guaranteed that the sample average of the mean in each leaf could be unbiased since the tree is developed on the sample data and each leaf only represents a small portion of it. Third, at the edge of each leaf, the regression estimates could experience a jump in the prediction value when moving across two leaf regions. This creates a discontinuity in the prediction that might be

undesired from the perspective of interpretation.

3 Bagging and Boosting

3.1 Bagging

Bootstrap Aggregating, as known as bagging, is an idea proposed by Breiman (1996) to improve the stability and accuracy of a learning procedure. Bagging creates a bootstrap sample by sampling from the population with replacement and generating a set of subsamples. Then for each subsample, the learning procedure is applied and the final prediction or classification is generating for all subsamples combined by averaging the output(regression) or voting(classification). Figure 2 represents a bagging classifier where classifiers are not identical across n. Each individual classifier takes a bootstrap sample for training and when a test sample is passed to the ensemble classifier, the classification prediction takes a majority vote from n individual classifiers. This example shows that bagging is an ensemble method besides being a learning method when each individual classifier takes the same learning algorithm.

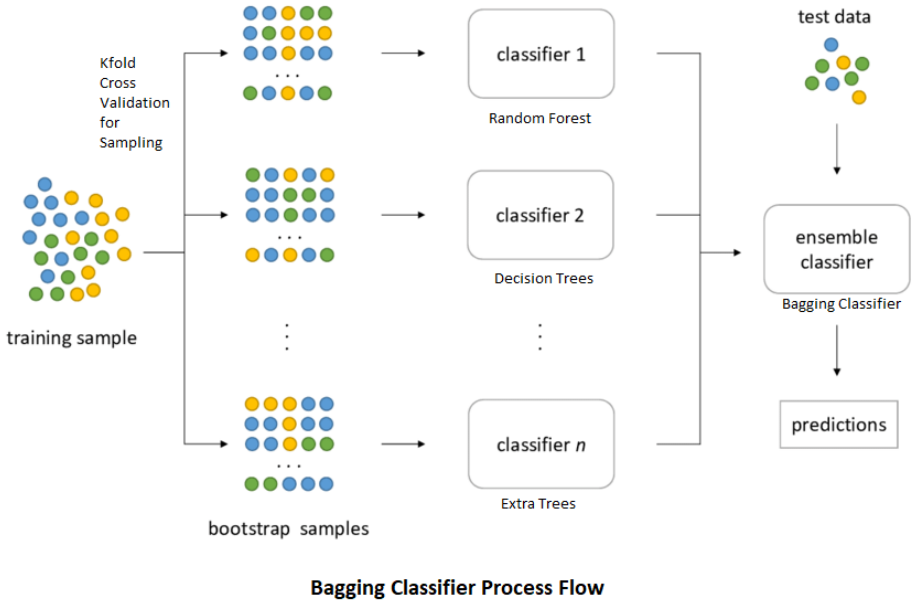

Figure 2: Illustration example for bagging. *Source*: https://medium.com/ml-research-lab/bagging-ensemble-meta-algorithm-for-reducing-variance-c98fffa5489f

Notice that bagging is not limited to the tree-based method but any learning method could use

bagging to generate an aggregated predictor. However, bagging brings significant improvement to a single-tree predictor because of its usage of bootstrap sampling. Recall that a regression tree might not provide an unbiased estimate since which tree structure and its resulting sample mean in each leaf is dependent on the training sample which could be not a good representation of the population distribution. Bootstrap sampling provides more training samples that represent different potential distributions of the population, thus taking the average over bootstrap samples could result in a prediction that is more immune to the sampling errors in the training data. Overall, bagging could improve the accuracy of learning procedures that performs unstably from different training samples and provides smooth and robust predictors.

3.2 Boosting

Boosting is a conceptually similar idea with bagging, which aims to enhance the performance of a weak learning algorithm, proposed firstly by Schapire (1990), Freund (1995), and Freund and Schapire (1997). The initial idea in Schapire (1990) aims to show that a weak learning algorithm could always be boosted into a strong learning one by learning specifically on the misclassified data for several extra rounds and taking the majority vote from all model outputs. This concept is carried over to later boosting methods, where for each round of prediction, extra weight will be assigned to the misclassified samples and the next round of training will be implemented on the reweighted sample. The final prediction will be a linear combination of the trained models from each stage.

Figure 3 represents the idea of Schapire (1990) with a boosted classification example. After each round of training, the misclassified samples are circled out and assigned higher weights in the next round's training sample, represented by the larger size of plus or minus signs. After three rounds of training, the final classifier adapt splitting rules from all three rounds and separate the sample space into 6 regions. The color of each region represents the classification decision from a majority vote of the three classifiers and the final prediction achieves higher accuracy than any of the previous three.

While there is a common view that explains boosting in terms of a "weighted majority vote" or "weighted committee", Friedman, Hastie and Tibshirani (2000) argue that this understanding might lead to some of the mystery about how and why the method works. Instead, viewing boosting as a technique for fitting an additive model could show more insights from the statistical perspective.

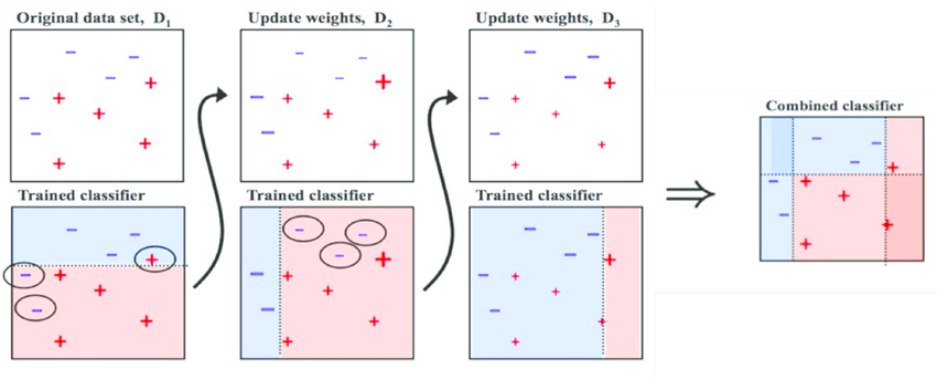

Figure 3: Illustration example for boosting. *Source*: https://medium.com/ml-research-lab/boosting-ensemble-meta-algorithm-for-reducing-bias-5b8bfdce281

For the regression problem, an additive model to estimate the mean $E(y|x) = F(x)$ has the form

$$F(x) = \sum_{j=1}^{p} f_j(x_j)$$

where each function $f_j(x_j)$ is a separate function for the input variables x_j, and x_j is a subset of the input variables. A backfitting update is performed to find the function $f_j(x_j)$ that minimizes the gap between the conditional expectation of the additive model without $f_j(x_j)$ and the true value

$$f_j(x_j) \longleftarrow E\Big[y - \sum_{k \neq j} f_k(x_k)|x_j\Big] \text{ for } j = 1, 2, ..., p, 1, 2, ...$$

The iteration keeps running until the model's prediction converges. We could extend the additive model to the set of functions, $\{\beta_m f_m(x; \gamma_m)\}_1^M$, which take all feature variables x as the input and are characterized by a set of parameters γ and a multiplier β_m. Then the additive model becomes

$$F_M(x) = \sum_{m=1}^{M} \beta_m f(x; \gamma_m)$$

One can find the optimal parameters $\{\beta_m, \gamma_m\}$ by solving the optimization problem

$$\{\beta_m, \gamma_m\} \longleftarrow \arg\min_{\beta,\gamma} L(y - \sum_{k \neq m}^{M} \beta_k f(x; \gamma_k) - \beta f(x; \gamma))$$

where $L(\cdot)$ is the loss function for the learning problem.

Now, if we consider the boosting procedure as finding the optimal learning function $f_m(x; \gamma_m)$ at each boosting round m, and finally the aggregate predictor is taking an average of all $\{f_m(x; \gamma_m)\}$ with the weight $\{\beta_m\}$, the boosted learning procedure results in the same prediction function that we solve from an additive model. This sketch proof shows that any boosting algorithm could be

viewed as some version of an additive prediction model. Zhang and Yu (2005) further shows that, with constructing boosting as an additive model, we could prove its numerical convergence when the greedy iteration increases and consistency with early stopping when the training sample size gets large for general loss functions (boosting forever can overfit the data, it is necessary to stop the boosting procedure early). In the next section, I will introduce two boosting algorithms from the two perspectives of understanding boosting mentioned previously: AdaBoost, a straightforward algorithm that applies recursive boosting, and Gradient Boosting, an alternative algorithm that optimizes boosted functions from the additive model view.

4 AdaBoost and Gradient Boosting

4.1 AdaBoost

This section will explain specifically the algorithm of AdaBoost and Gradient Boosting. The idea of AdaBoost is mentioned in the last section: for each iteration, the algorithm fits the model with the weighted training data by minimizing a certain loss function. Then the fitted data will provide an adjustment to the weight distribution by assigning higher weights to the bad fitted samples for the use of next iteration. The final output will be a weighted aggregation of the model from each iteration. One version of AdaBoost procedure for binary classification problem is the Discrete AdaBoost (Freund and Schapire 1996), as shown in Figure 4. The dependent variable y has values 1 or -1, and we define our boosted model $F(x) = \sum_1^M c_m f_m(x)$ to predict the classification of y given the feature data x. Specifically, each $f_m(x)$ is a classifier producing values either 1 or -1, c_m are constants and the classification decision of the boosted model on y is $\text{sign}(F(x))$, namely whether the aggregated value of M classifiers is positive or negative.

Discrete AdaBoost [Freund and Schapire (1996b)]

1. Start with weights $w_i = 1/N, i = 1, \ldots, N$.
2. Repeat for $m = 1, 2, \ldots, M$:
   (a) Fit the classifier $f_m(x) \in \{-1, 1\}$ using weights w_i on the training data.
   (b) Compute $\text{err}_m = E_w[1_{(y \neq f_m(x))}]$, $c_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (c) Set $w_i \leftarrow w_i \exp[c_m 1_{(y_i \neq f_m(x_i))}]$, $i = 1, 2, \ldots, N$, and renormalize so that $\sum_i w_i = 1$.
3. Output the classifier $\text{sign}[\sum_{m=1}^M c_m f_m(x)]$.

Figure 4: Discrete AdaBoost Algorithm. *Source*: Friedman, Hastie and Tibshirani (2000)

The Discrete AdaBoost trains the classifiers $f_m(x)$ on a weighted training sample, with weight w_i for training sample i. After model $f_m(x)$ is trained, the weighted error rate is calculated by $\text{err}_m = E_w[1_(y \neq f_m(x)]$, the expected rate of misclassification on the weighted training sample. Then a parameter to measure the importance of the classifier $f_m(x)$ within the whole model, $c_m = \log((1 - \text{err}_m)/\text{err}_m)$ is a logit transformation of the weighted accuracy rate $1 - \text{err}_m$. As we could only boost a weak learner into a strong learner, it is required that the classifier performs at least better than random guessing, that is $E[1_(y \neq f_m(x)] > 0.5$, thus $c_m \in [0, +\infty)$ measures the accuracy of the classifier $f_m(x)$ and the weight it should occupy in the aggregate model. Also, the adjusted weight for the next round of training is imposed on the misclassified samples with the multiplier c_m and renormalized to have a unit weight in total. After M iterations, the boosted classifier outputs $\text{sign}[\sum_{m=1}^{M} c_m f_m(x)]$ as the classification decision. With the Discrete AdaBoost example, we could observe the reason that boosting is interpreted as a "weighted majority vote" or "weighted committee" as it takes the weighted aggregate predictions from the individual classifiers to produce the final decision.

4.2 Gradient Boosting

Gradient Boosting (Friedman 1999) reforms the boosting procedure by optimizing an additive model, where each iteration adds a new prediction function to optimize the prediction of the sum of all previous functions. In this way, the optimal incremental function is constructed by the function that provides the steepest-descent gradient in the functional space for the overall prediction function. The key idea is to obtain an estimate or approximation $\hat{F}(x)$, of the function $F^*(x)$ that minimizes the expected loss function of the fitted model

$$F^* = \arg\min_F E_x[E_y(L(y, F(x)))|x]$$

and we specify the form of the fitting function $F(x)$ to be the additive model mentioned in section 3.2

$$F_M(x) = \sum_{m=1}^{M} \beta_m f(x; \gamma_m).$$

To find the solution of F^*, one of the frequently used numerical minimization methods, steepest-descent could be useful. In general, we seek to minimize a model's expected loss function

$$\Phi(F(x)) = E_y L[y, F(x)|x]$$

with respective to $F(x)$ and our solution is $F^*(x) = \sum_{m=0}^{M} f_m(x)$, an additive model, where $f_0(x)$ is an initial guess and $\{f_m(x)\}_1^M$ are incremental functions ("steps" or "boosts") searched by steepest-

descent algorithm

$$f_m(x) = -\rho_m g_m(x)$$

with gradient vector at m step

$$g_m(x) = \Big[\frac{\partial \Phi(F(x))}{\partial F(x)}\Big]_{F(x)=F_{m-1}(x)}$$

and

$$F_{m-1}(x) = \sum_{i=0}^{m-1} f_i(x).$$

The multiplier ρ_m is given by

$$\rho_m = \arg\min_{\rho} E_{y,x} L(y, F_{m-1}(x) - \rho g_m(x)).$$

The interpretation of steepest-descent optimization is intuitive: we seek to reach the global minimum on the hyperplane of the loss function in a functional space, while steepest-descent aims to find it with a step-by-step procedure. We start at an initial guess location, and for every step, we look for the function that could provide the steepest descent gradient on the hyperplane, essentially it is the direction that provides the largest reduction on the loss function. If the loss function gives a convex hyperplane, then steepest-descent could guarantee the global minimum within infinitely small error. In conclusion, by viewing boosting as an additive model, we could apply the steepest-descent procedure to find a general form of the "boost", or the incremental functions that minimize the loss function of the aggregate model in each iteration.

5 RandomForest

While boosting methods aim to enhance the performance of some learning procedure, especially a single tree model, by recursively applying the procedure on the training sample one iteration after another one, we could also consider to build many trees at the same time and aggregate their outputs to produce the prediction, which is the random forest method (Ho 1995, Breiman 2001). The first random decision forest algorithm proposed by Ho (1995) builds trees that are trained by a random subspace method: each tree is trained on a randomly picked subset of the feature variables. The random drop on the features forces the whole forest not to depend its prediction on any individual feature that performs well in prediction for the training samples but potentially not robust to other samples. The later extension RandomForest by Breiman (2001) keeps the randomness in features but applies it at node level combined with a bagging procedure. Specifically, in the growth of each

tree, RandomForest does not assign a fixed subset of features to the tree, but when the tree makes a split at a node, a random subset of features are given for the split decision. This node level feature selection brings more randomness in the variable dependency, so that not only the whole forest, but every single tree could not produce estimates highly dependent on certain well-performing features. RandomForest also applies the bagging procedure in the sample data used to train trees, so that each tree is fitting a different bootstrap sample as well as the implicit data distribution. This bagging procedure avoids the shortcoming that a tree model's structure is highly dependent on the sample data and thus its performance is not robust to changes in training samples. Overall, random forest method is the horizontal ensemble of many trees that are independently grown based on the random training data.

Athey et al. (2018) propose the generalized random forests that extend the application of random forests to fit a more general set of learning objects with known asymptotic distributions of the estimates. Instead of training the model to fit the object variable, generalized random forests seek to estimate a set of parameters by minimizing the local moment equations adjusted by a similarity weight. This approach is similar to the non-parametric kernel regression which uses a kernel function to weight the neighborhood points to calculate for the weighted local average as the estimates. Instead of deterministic weight functions, Athey et al. (2018) use random forests to measure the similarity score between two sample points, that is the frequency of two training data points fall into the same leaf. Besides the typical procedures of RandomForest, generalized random forest considers a splitting rule that maximizes the heterogeneity between the two child nodes. With this specific setup, they develop a large sample theory showing that the generalized random forests estimates are consistent and asymptotically Gaussian and they also provide an estimator for the asymptotic variance that enables valid confidence intervals.

6 XGBoost

XGBoost (Chen and Guestrin, 2016) adopts the gradient boosted tree algorithm and performs other machine learning techniques to improve its performance and robustness. XGBoost first builds its trees by gradient boosting with a regularized objective function

$$L(y, F(x)) = l(y, F_m(x)) + \sum_m \Omega(f_m)$$

where $l(y, F_m(x))$ is some standard convex loss function, $\Omega(f_m) = \gamma T + \frac{1}{2}\lambda||\omega||^2$ is a penalty on the complexity of the tree structure, T is the number of leaves in the tree and ω is the vector of leaf weights. Then XGBoost seeks the function $f_m(x)$ to minimize the regularized objective function

$$L^{(m)} = L(y, F_{m-1}(x) + f_m(x)) + \Omega(f_m)$$

by performing the gradient descent algorithm to find the optimal tree structure f_m. The regularized objective leads the model to select a tree that is simpler rather than one with many leaves that could result in overfitting.

Besides the regularized objective, XGBoost also applies shrinkage and the feature selection to prevent overfitting. Shrinkage is used to scale down the weights of each new boosted tree in the aggregate model by a fixed proportion. Such shrinkage reduces the influence of each individual tree and leaves more improvement space for future trees. Feature selection, the key idea in random forest methods, is also implemented in the process of making splits. Only a random subset of features is available for splitting at each node during the growth of the tree.

Sparsity-aware split finding is an innovation of XGBoost in the field of tree-related methods, as it could handle all patterns of sparse data and perform the algorithm in an efficient way. The commonly used greedy algorithm for split finding is very powerful since it could compare all possible splitting candidates to find the optimal one, however, it is also computationally costly as it needs to search for almost every point in the training set. One alternative way to save the computational timing is to propose some candidate splitting points according to percentiles of feature distribution(e.g. quantile points) and the algorithm only picks the optimal splits among the candidates. However, this approach is limited to the dense data as for categorical features, there could be only few input values and the proposed candidate split points based on distribution could be missing in the training data. XGBoost handles this sparsity of the feature by assigning default direction for the missing values in the proposed candidates, and the default direction is learned from the data so that it could assign a close non-missing value that maximizes the model performance to replace the missing candidates. In this way, XGBoost could cope with all sparsity patterns in a unified way.

In brief, XGBoost combines many techniques that are shown to be effective for other machine learning tasks into the gradient boosted tree method, and fine tunes the candidate proposal procedure by its innovative sparsity-aware split finding algorithm. The experimental results displayed in the paper show that XGBoost significantly reduces the computational timing needed for the

learning task while remains a competent prediction accuracy compared to other top performing machine learning methods.

7 Generalized Random Forests

This section aims to elaborate the work in Susan et al (2018) from following three perspectives. First, the theoretical model and the environment of the problem and the method this paper proposes. Then I perform a set of simulation work to display what statistical tasks would benefit the most from generalized random forests and what may not be suitable for it. Finally, I go over one simple empirical work with the usage of generalized random forests.

7.1 Theoretical Model

In this section, I will briefly cover the theoretical model and innovation of the generalized random forests method without mentioning too many technical details. The major goal is to understand what steps of statistical learning are involved and grasp the essence of the algorithm in a intuitional way. Before we dive into the generalized random forests, let us start from scratch. In the "vanilla" version of the regression forests (Breiman, 2001), individual trees are grown by greedy recursive partitioning such that we recursively add axis-aligned splits to the tree, where each split aims to maximize the improvement to model fit. Figure 1 is a graph illustration on how regression forests decide the splitting points. To generate one regression tree for a data sample with size $n = 7$, its covariate matrix X has two dimensions, we randomly pick one of the two dimensions, X_1 in the example, and find the splitting point at $X_1 = 0.3$ that improves the model fit the most from separating the sample into two subsets. We call the two subsets "child nodes" and for the set before separating we call it "parent node". After we generate the child nodes, we could treat each child node as a new parent node and repeat what we have done so far, that is optimally splitting the set into two child nodes over one randomly picked dimension. In the example of Figure 1, we have 4 child nodes at the end, and we call such end nodes "leaves". Finally, as every data point x could be classified by the splitting points at each stage so it must fall in one leaf, the regression tree's output of predicted value $\hat{Y}$ is simply the average value of Y in the leaf, $\bar{Y}$, that a data point x falls in. While this prediction is made by only one tree based on a small sample, we could perform bootstrapped sampling for N samples and generate one tree per sample, thus we will have N trees for predicting the data point x. Our final prediction based on the forest of trees, $\{Y_i\}_{i=1,\dots,N}$, is

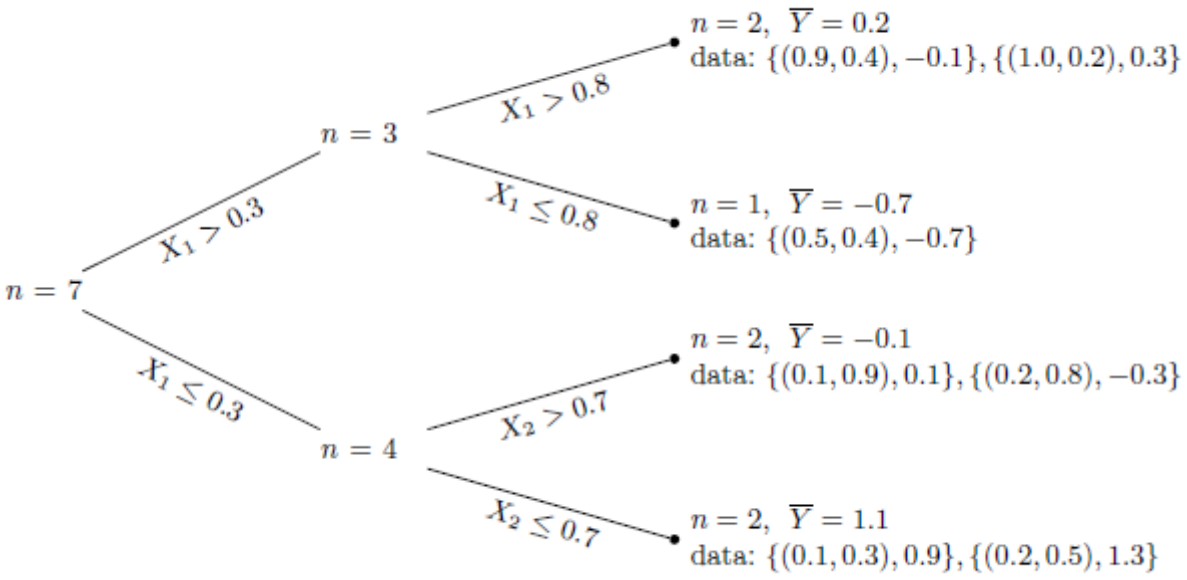

Figure 5: Regression Forests (from Susan et al (2018))

simply the averaged value of tree predictions, $\frac{1}{N}\sum_{i=1}^{N} Y_i$.

In generalized random forests, it preserve the core elements of the forests method – including recursive partitioning, subsampling, and random split selection – but abandon the idea that our final estimate is obtained by averaging estimates from each member of an ensemble. Instead, it treats forests as a type of adaptive nearest neighbor estimator, that I will elaborate further in the following.

7.1.1 Forest-Based Local Estimation

The environment contains n independent and identically distributed samples indexed by $i = 1, ..., n$, an observable quantity O_i for each sample that we wants to estimate by $\theta(\cdot)$, and a set of auxiliary covariates X_i. For regression tasks, the observable just consists of an outcome $O_i = \{Y_i\}$; in the case of treatment effect estimation with exogenous treatment assignment, it also includes the treatment assignment $O_i = \{Y_i, W_i\}$. Our goal is to estimate solutions to local estimation equations of the form $\mathbb{E}[\psi_{\theta(x)}(O_i)|X_i = x] = 0$ for all $x \in \mathcal{X}$. The approach of generalized random forests is to first define a similarity weights $\alpha_i(x)$ that assign higher weights to i-th sample if it is

more relevant in the process of fitting $\theta(\cdot)$ at x, and then fit to a overall estimating equation:

$$\hat{\theta}(x) \in \arg\min_{\theta} \left\{ \left\| \sum_{t=1}^{n}] \alpha_i(x) \psi_\theta(O_i) \right\|_2 \right\}. \tag{1}$$

It is noteworthy that if the weights $\alpha_i(x)$ is some deterministic kernel function, then the equation (1) simply generates our traditional non-parametric estimator, which works well in low dimensions but sensitive to the curse of dimensionality. Here, weights are obtained by averaging neighborhoods implicitly produced by different trees, that is the frequency of i-th training example falls into the same leaf as x:

$$\alpha_{bi}(x) = \frac{\mathbf{1}(\{X_i \in L_b(x)\})}{|L_b(x)|}, \alpha_i(x) = \frac{1}{B} \sum_{b=1}^{B} \alpha_{bi}(x) \tag{2}$$

where we have trees indexed by $b = 1, ..., B$ and define the leaf that x falls in tree b as $L_b(x)$. Figure

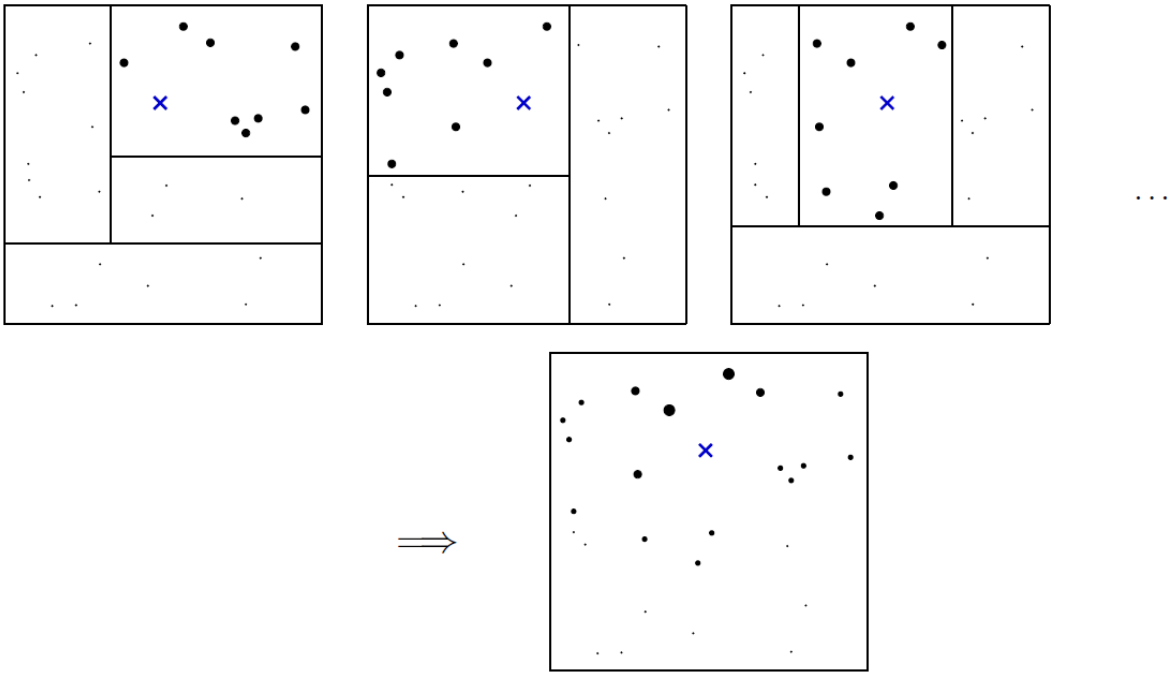

Figure 6: random forest weighting function(from Susan et al (2018))

2 is the graph illustration of the random forest weighting function. The top three diagrams display different splitting results from three trees generated from some random sampling data, where only the dots that fall into the same leaf with our target x are marked out. When we combine three trees into the bottom diagram, we will assign larger sizes to the points that have higher frequencies being marked in previous diagrams, so the points fall in the same leafs with x three times have the largest size, and thus points with larger size will have larger weight when we fit equation (1) locally

at x. With the weights given by equation (2), we could solve equation (1) to have our estimator $\hat{\theta}(x)$; the only thing remains unclear is what splitting rule it will use to generate the forests that provides us the weighting function.

7.1.2 Splitting to Maximize Heterogeneity

As mentioned, the main difference between random forests relative to other non-parametric regression techniques is their use of recursive partitioning on subsamples to generate the weighting function $\alpha_i(x)$. Specifically, the splitting scheme used in generalized random forests focuses on heterogeneity in the target functional $\theta(X)$. To divide a parent node $P \subseteq \mathcal{X}$ into two children $C_1, C_2 \subseteq \mathcal{X}$ at some splitting point, generally we seek to minimize a loss function defined over a sample set $\mathcal{J}$ as $err(C_1, C_2) = \sum_{j=1.2} \mathbb{P}[X \in C_j | X \in P]\mathbb{E}[(\hat{\theta}_{C_j}(\mathcal{J}) - \theta(X))^2 | X \in C_j]$. In standard regression tree implementations, we might directly minimize the in-sample prediction error of the node or minimize the entropy after splitting to maximize the information gain. This direct loss minimization is not proper under current settings as the identification via a moment condition does not generally have a unbiased, model-free estimate of the loss term $err(C_1, C_2)$. To address this issue, under certain assumptions, Susan et al (2018) find a way to decompose the loss function into three parts such that $err(C_1, C_2) = K(P) - \mathbb{E}[\Delta(C_1, C_2)] + o(r^2)$ with definition

$$\Delta(C_1, C_2) := n_{C_1} n_{C_2} / n_P^2 \big(\hat{\theta}_{C_1}(\mathcal{J}) - \hat{\theta}_{C_2}(\mathcal{J})\big)^2, \tag{3}$$

where n_{C_1}, n_{C_2}, n_P are the numbers of observations in C_1, C_2, P respectively, $K(P)$ is a deterministic term that measures the purity of the parent node that does not depend on how the parent is split, the o-term depends on sampling variance and our target term $\Delta(C_1, C_2)$ captures the heterogeneity of the two children nodes. With this transformation, minimizing the loss function essentially needs to maximize the term in (3) and as a result, the splitting rule will increase the heterogeneity of the estimates as fast as possible.

While we have a good theoretical criterion to perform splits by maximizing (3), explicitly solving for $\hat{\theta}_{C_1}, \hat{\theta}_{C_2}$ in each candidate child by equation (1) is too expensive computationally. Authors also provide a gradient-based approximations for $\hat{\theta}_{C_1}, \hat{\theta}_{C_2}$ to avoid this issue. We first compute A_P as any consistent estimate for the gradient of the expectation of the ψ-function, i.e., $\nabla \mathbb{E}[\psi_{\hat{\theta}_P}(O_i) | X_i \in P]$, and then using a gradient descending algorithm to approximate

$$\tilde{\theta}_C = \hat{\theta}_P - \frac{1}{|\{i : X_i \in C\}|} \sum_{\{i : X_i \in C\}} A_P^{-1} \psi_{\hat{\theta}_P}(O_i), \tag{4}$$

Once we have the approximate estimates $\tilde{\theta}_{C_1}, \tilde{\theta}_{C_2}$, we could further construct approximate term for (3), $\tilde{\Delta}(C_1, C_2)$. To justify this approximation will not influence the consistency, authors provides the proof(which I will skip here) that if A_p is a consistent estimator for the gradient of expectation of the score function $\psi_{\hat{\theta}_P}(O_i)$, then $\tilde{\Delta}(C_1, C_2)$ equals $\Delta(C_1, C_2)$ plus some error term that converges to zero when sample is large enough. With all the elements of the generalize random forest explained so far, finally we could ensemble the algorithm together by the pseudo code in Figure 3.

7.2 Applications

Susan et el (2018) claims that their generalized random forest could help develop new methods for many statistical tasks including non-parametric quantile regression and conditional average partial effect estimation, thus I will run several simulations to exhibit the performance of generalized random forest over these tasks as well as other methods' performances.

7.2.1 Quantile Regression Forests

Non-parametric quantile regression is a classical problem that has been considered in detail by Meinshausen (2006) by a consistent forest-based quantile regression. The difference between Meinshausen (2006) and generalized random forest lies in the splitting rule, such that Meinshausen (2006) uses a plain CART regression splits while this method utilizes the criterion that is designed to capture heterogeneity in conditional quantiles. As a benchmark, I also compare these two forest based method with the traditional kernel based method. The result is shown in Figure 4.

The simulation generates a sample with 2000 observations where X_i is uniformly distributed over $[-1, 1]^p$ with $p = 40$ and Y_i is Gaussian conditionally on $(X_i)_1$. In the top panel, we have a mean shift in the distribution of Y_i conditional on X_i at $(X_i)_1 = 0$, and all three methods are able to pick up the change while kernel regression generates more fluctuations in the predictions relative to the forest methods. However, in the bottom panel, while the conditional mean of Y is constant, but there is a scale shift at $(X_i)_1 = 0$. In this case, generalized random forest performs still well as its splitting scheme could capture the heterogeneity in the sample space, while other two methods could only update their predictions slowly as they are only sensitive to changes in the conditional mean of Y given X. Specifically, kernel regression could slowly get to the correct level when the local neighborhood's heterogeneity gradually disappears but quantile regression forest completely broke down as its global splitting rule could not capture the changes in the quantiles at all and

Algorithm 1 Generalized random forest with honesty and subsampling

All tuning parameters are pre-specified, including the number of trees B and the sub-sampling s rate used in SUBSAMPLE. This function is implemented in the package `grf` for `R` and `C++`.

procedure GENERALIZEDRANDOMFOREST(set of examples $\mathcal{S}$, test point x)
  weight vector $\alpha \leftarrow$ ZEROS($|\mathcal{S}|$)
  for $b = 1$ to total number of trees B **do**
    set of examples $\mathcal{I} \leftarrow$ SUBSAMPLE($\mathcal{S}$, s)
    sets of examples $\mathcal{J}_1$, $\mathcal{J}_2 \leftarrow$ SPLITSAMPLE($\mathcal{I}$)
    tree $\mathcal{T} \leftarrow$ GRADIENTTREE($\mathcal{J}_1$, $\mathcal{X}$) ▷ See Algorithm 2.
    $\mathcal{N} \leftarrow$NEIGHBORS(x, $\mathcal{T}$, $\mathcal{J}_2$) ▷ Returns those elements of $\mathcal{J}_2$ that fall into the same leaf as x in the tree $\mathcal{T}$.
    for all example $e \in \mathcal{N}$ **do**
      $\alpha[e]$ += $1/|\mathcal{N}|$
  output $\hat{\theta}(x)$, the solution to (2) with weights α/B

The function ZEROS creates a vector of zeros of length $|\mathcal{S}|$; SUBSAMPLE draws a subsample of size s from $\mathcal{S}$ without replacement; and SPLITSAMPLE randomly divides a set into two evenly-sized, non-overlapping halves. The step (2) can be solved using any numerical estimator. Our implementation `grf` provides an explicit plug-in point where a user can write a solver for (2) appropriate for their ψ-function. $\mathcal{X}$ is the domain of the X_i. In our analysis, we consider a restricted class of generalized random forests satisfying Specification 1.

Algorithm 2 Gradient tree

Gradient trees are grown as subroutines of a generalized random forest.

procedure GRADIENTTREE(set of examples $\mathcal{J}$, domain $\mathcal{X}$)
  node $P_0 \leftarrow$ CREATENODE($\mathcal{J}$, $\mathcal{X}$)
  queue $\mathcal{Q} \leftarrow$ INITIALIZEQUEUE(P_0)
  while NOTNULL(node $P \leftarrow$ POP($\mathcal{Q}$)) **do**
    $(\hat{\theta}_P, \hat{\nu}_P, A_P) \leftarrow$ SOLVEESTIMATINGEQUATION(P) ▷ Computes (4) and (7).
    vector $R_P \leftarrow$ GETPSEUDOOUTCOMES($\hat{\theta}_P$, $\hat{\nu}_P$, A_P) ▷ Applies (8) over P.
    split $\Sigma \leftarrow$ MAKECARTSPLIT(P, R_P) ▷ Optimizes (9).
    if SPLITSUCCEEDED(Σ) **then**
      SETCHILDREN(P, GETLEFTCHILD(Σ), GETRIGHTCHILD(Σ))
      ADDTOQUEUE($\mathcal{Q}$, GETLEFTCHILD(Σ))
      ADDTOQUEUE($\mathcal{Q}$, GETRIGHTCHILD(Σ))
  output tree with root node P_0

The function call INITIALIZEQUEUE initializes a queue with a single element; POP returns and removes the oldest element of a queue $\mathcal{Q}$, unless $\mathcal{Q}$ is empty in which case it returns null. MAKECARTSPLIT runs a CART split on the pseudo-outcomes, and either returns two child nodes or a failure message that no legal split is possible.

Figure 7: Generalized random forest Algorithms (from Susan et al (2018))

consistently produced a biased estimation.

One of the challenges with generalized random forests could exhibit edge effects whereby the the estimation could taper off as we approach the edge of $\mathcal{X}$-space, as the splitting scheme could not assign large enough weights to the limit number of observations at the edge. My next simulation replicate this special case and want to check what could happen to the three methods when the data could observe an edge effect. The simulation structure is identical to the previous one, except the mean shift happens at $(X_i)_1 = -0.9$ and the scale shift happens at $(X_i)_1 = -0.8$, the results are shown in Figure 5. Not surprisingly, two forest methods could not adapt correctly at the edge of the sample space and their predictions are generally off target, while kernel quantile regression could still update its local neighborhood regardless the sparse data at the edge, thus kernel method is more robust to the edge effect relative to the forest methods.

7.2.2 Conditional Average Treatment Effect Estimation

Next, we could consider conditional average treatment effect estimation under exogeneity. Now we observe samples (X_i, Y_i, W_i) where W_i is the exogenous treatment. We have a random effect model $Y_i = W_i \cdot b_i + \varepsilon_i, \beta(x) = \mathbb{E}[b_i | X_i = x]$ is the conditional average treatment effect (CATE) of our interest, and this problem is equivalent to the heterogeneous treatment effect estimation under unconfoundedness under this setup. Again we simulate a sample with 2000 observations where X_i is uniformly distributed over $[-1, 1]^p$ with $p = 10$ but now Y_i depends on both the treatment effect from W_i and some transformation of X_i. Moreover, treatment W_i has a linear increasing effect over $(X_i)_1$ conditional on $(X_i)_1 > 0$. Our goal is to capture the heterogeneous treatment effect which is shown in Figure 6. We could see the estimated CATE is closely following the heterogeneity due to the covariate variable while the 95% confidence interval constructed by the large sample variance estimator (inference not covered in this note) also contains the true value all the time.

7.3 Empirical Example

Finally, I perform the generalized random forest on the research project of Koop and Tobias (2004) which originally investigated the heterogeneous effect in returns to schooling. Following their idea, we want to check whether return to schooling is influenced by other factors such as experience, family background, etc. To consider both continuous and discrete treatments, two approaches are taken as one treats year of education *edu* as a continuous variable and the another creates a dummy

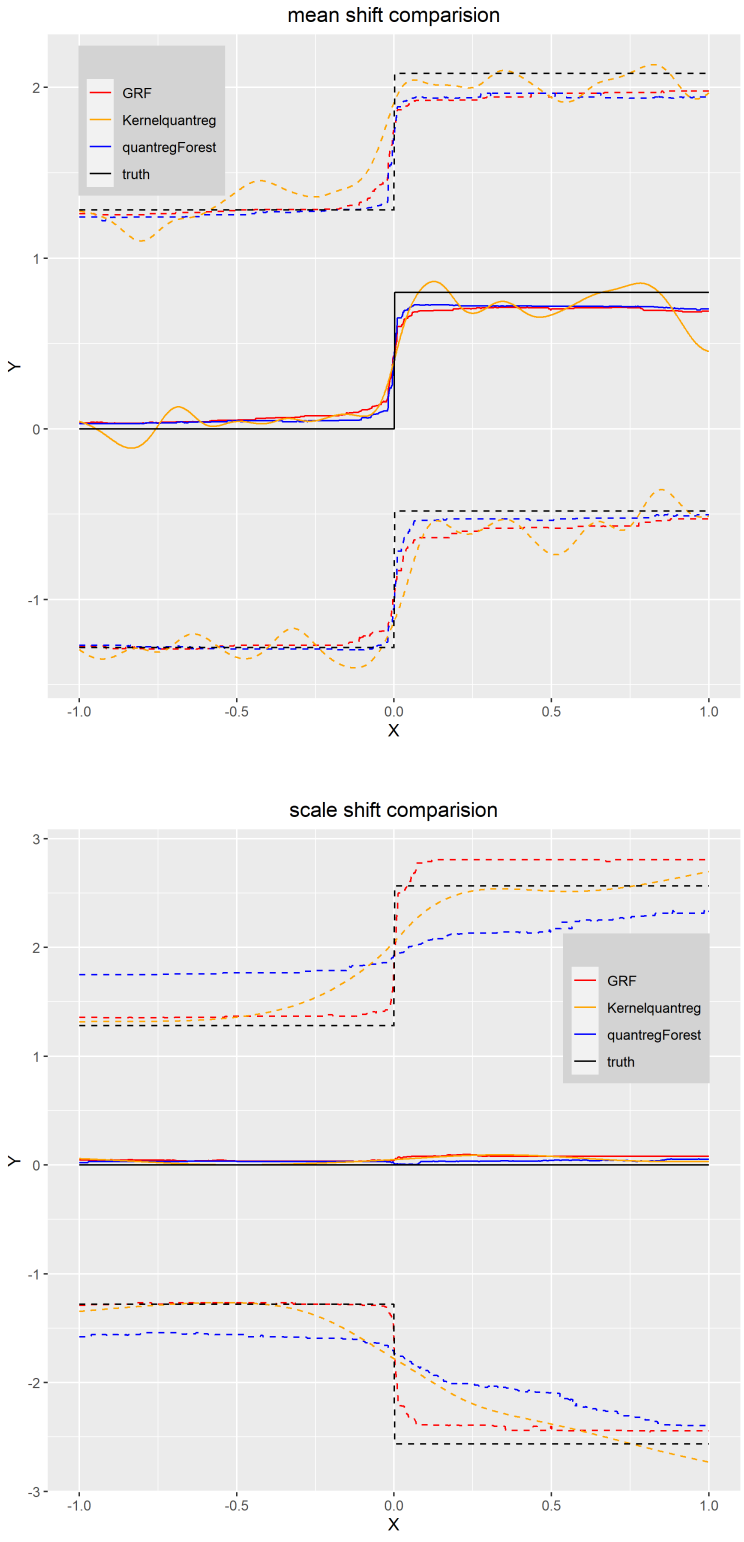

Figure 8: Comparison of quantile regression using generalized random forests, Meinshause (2006) and kernel regression

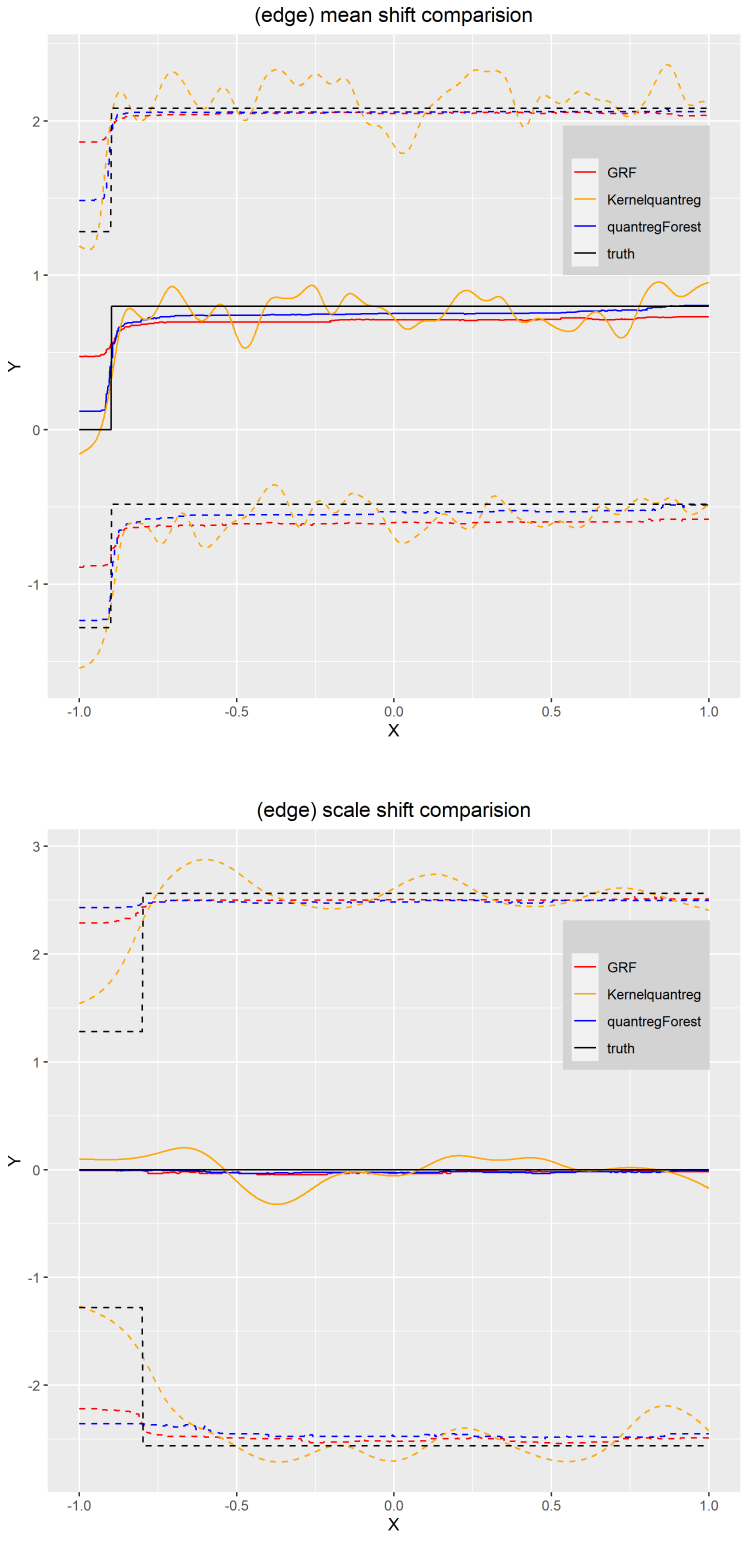

Figure 9: Comparison of quantile regression with edge effect using generalized random forests, Meinshause (2006) and kernel regression

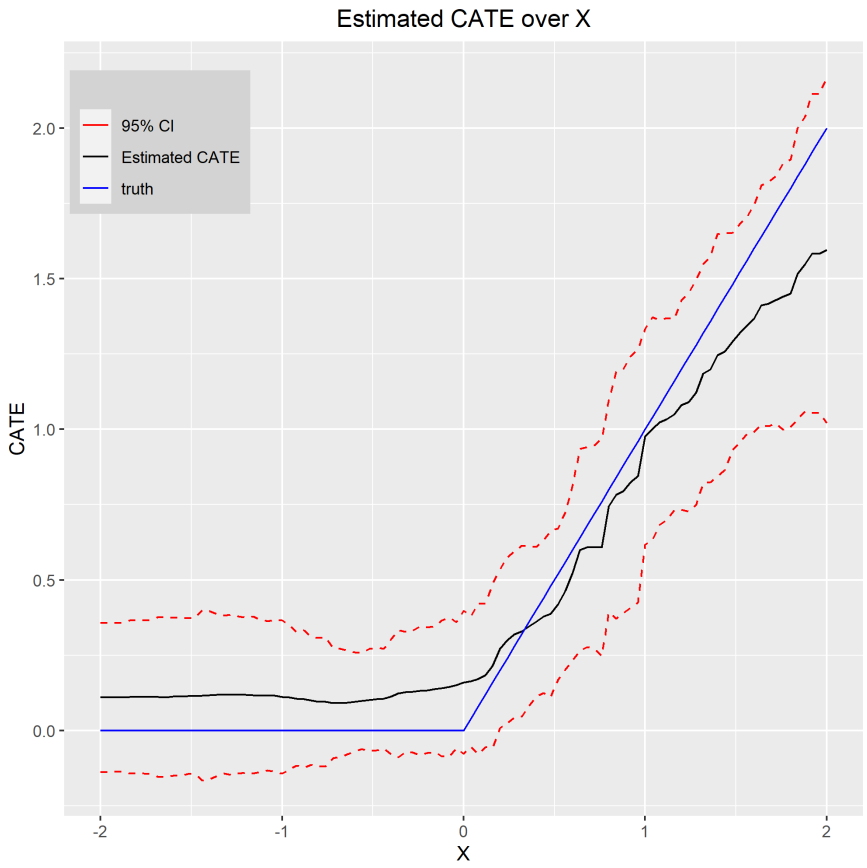

Figure 10: heterogeneous treatment effect by generalized random forest

for whether individual attends high school or not, or equivalently $\mathbf{1}(edu > 10)$.

With these two approaches, we are essentially considering conditional average partial effect (CAPE) and conditional average treatment effect (CATE) of return to schooling, respectively. To perform the test of heterogeneity by generalized random forest, we need to estimate CAPE or CATE over one covariate variable at a time, while due to limit of pages, I will primarily present the results for the analysis for heterogeneity of work experience, which are shown in Figure 7 & 8.

In both approaches, we could observe certain heterogeneity on returns to schooling over work experience but not to a significant level. The OLS estimated value falls in the 95% confidence interval for most of the range. Specifically, we could observe CAPE has a small positive trend as experience increases while CATE has a less significant change. This corresponds to the result of Koop and Tobias (2004) as they claim the heterogeneity is continuous rather than discrete. When we force the treatment effect to be estimated at a certain threshold, we break the continuity of the sample space such that CATE could not capture the more complex heterogeneity that could be

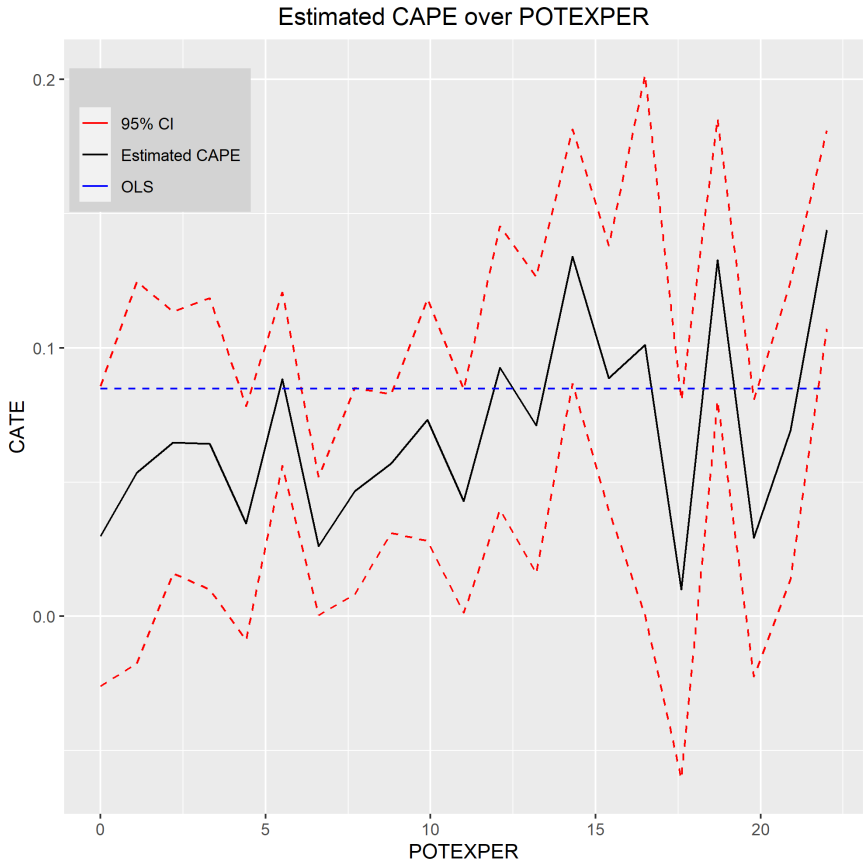

Figure 11: heterogeneous partial effect by generalized random forest

generated by factors across the threshold. Another potential reason could be the current setup of generalized random forest is not proper to deal with confounded treatment effect, while in the case of return to schooling, confoundedness is reasonable to exist so that the heterogeneity analysis over only one factor is not valid.

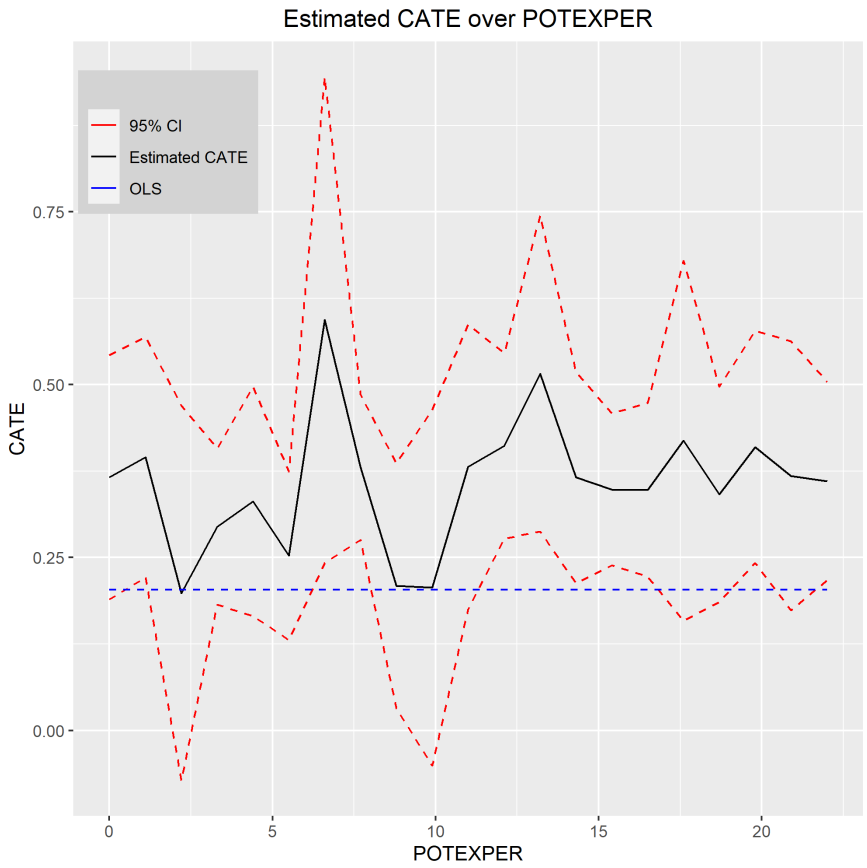

Figure 12: heterogeneous treatment effect by generalized random forest

8 Conclusion

In this piece of literature review, I try to cover the development of tree-based machine learning methods from the original decision tree to the recently developed XGBoost. Almost every component of XGBoost represents a crucial idea or technique in the field of machine learning or statistics, thus it provides a good chance to backtrack the evolution of many important machine learning methods, such as boosting, gradient boosting and random forest.

I also cover the the essence of generalized random forest proposed by Susan et al (2018). Conceptual steps about how to perform generalized random forest are elaborated via graph illustration and some intuitive and normative way, while a large set of contents about technical details (including its asymptotic property and inference of confidence interval) are skipped. Several simulation showcases in quantile regression and heterogeneous treatment effect are displayed to show the performance of generalized random forest on these specific statistical learning tasks with comparisons of other traditional methods.

From the econometric perspective, understanding the theoretical background of machine learning methods and their asymptotic properties also sheds light to the inference tasks involved in economic researches. A model that fits the dependent variable nicely without hurting the consistency of the estimators would provide much more nuisances in the variance of data and improve the interpretability of the economic model by a large step.

References

Susan Athey and Guido W. Imbens. Machine Learning Methods That Economists Should Know About. *Annual Review of Economics*, 11(1):685–725, August 2019. ISSN 1941-1383, 1941-1391. doi: 10.1146/annurev-economics-080217-053433. URL https://www.annualreviews.org/doi/10.1146/annurev-economics-080217-053433.

Susan Athey, Julie Tibshirani, and Stefan Wager. Generalized Random Forests. *arXiv:1610.01271 [econ, stat]*, April 2018. URL http://arxiv.org/abs/1610.01271. arXiv: 1610.01271.

Leo Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, August 1996. ISSN 1573-0565. doi: 10.1023/A:1018054314350. URL https://doi.org/10.1023/A:1018054314350.

Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, October 2001. ISSN 1573-0565. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

Leo Breiman, Jerome Friedman, Charles J. Stone, and R. A. Olshen. *Classification and Regression Trees*. Taylor & Francis, January 1984. ISBN 978-0-412-04841-8.

Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, August 2016. doi: 10.1145/2939672.2939785. URL http://arxiv.org/abs/1603.02754. arXiv: 1603.02754.

Y. Freund. Boosting a Weak Learning Algorithm by Majority. *Information and Computation*, 121(2):256–285, September 1995. ISSN 0890-5401. doi: 10.1006/inco.1995.1136. URL http://www.sciencedirect.com/science/article/pii/S0890540185711364.

Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*, ICML'96, pages 148–156, Bari, Italy, July 1996. Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-419-3.

Yoav Freund and Robert E Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55 (1):119–139, August 1997. ISSN 0022-0000. doi: 10.1006/jcss.1997.1504. URL http://www.sciencedirect.com/science/article/pii/S002200009791504X.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. ADDITIVE LOGISTIC REGRESSION: A STATISTICAL VIEW OF BOOSTING. page 71.

Jerome H Friedman. GREEDY FUNCTION APPROXIMATION: A GRADIENT BOOSTING MACHINE. page 44.

Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1, August 1995. doi: 10.1109/ICDAR.1995.598994.

Gary Koop and Justin L. Tobias. Learning about heterogeneity in returns to schooling. *Journal of Applied Econometrics*, 19(7):827–849, 2004. ISSN 1099-1255. doi:

10.1002/jae.744. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/jae.744. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jae.744.

Nicolai Meinshausen. Quantile Regression Forests. *Journal of Machine Learning Research*, 7(Jun):983–999, 2006. ISSN ISSN 1533-7928. URL http://www.jmlr.org/papers/v7/meinshausen06a.html.

James N. Morgan and John A. Sonquist. Problems in the Analysis of Survey Data, and a Proposal. *Journal of the American Statistical Association*, 58(302):415–434, June 1963. ISSN 0162-1459. doi: 10.1080/01621459.1963.10500855. URL https://amstat.tandfonline.com/doi/abs/10.1080/01621459.1963.10500855. Publisher: Taylor & Francis.

Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, June 1990. ISSN 1573-0565. doi: 10.1007/BF00116037. URL https://doi.org/10.1007/BF00116037.

Tong Zhang and Bin Yu. Boosting with early stopping: Convergence and consistency. *The Annals of Statistics*, 33(4):1538–1579, August 2005. ISSN 0090-5364. doi: 10.1214/009053605000000255. URL https://projecteuclid.org/euclid.aos/1123250222.