Generalized Random Forests

Generalized Random Forests

The main competitor to double/debiased machine learning is **generalized**, or **causal**, random forests. The key papers are:

Generalized Random Forests

The main competitor to double/debiased machine learning is **generalized**, or **causal**, random forests. The key papers are:

Susan Athey and Guido Imbens (2016), "Recursive partitioning for heterogeneous causal effects," *Proceedings of the National Academy of Sciences*, 113, 7353–7360. (2213 cites)

Generalized Random Forests

The main competitor to double/debiased machine learning is **generalized**, or **causal**, random forests. The key papers are:

Susan Athey and Guido Imbens (2016), "Recursive partitioning for heterogeneous causal effects," *Proceedings of the National Academy of Sciences*, 113, 7353–7360. (2213 cites)

Stefan Wager and Susan Athey (2018), "Estimation and inference of heterogeneous treatment effects using random forests," *JASA*, 113, 1228–1242. (3681 cites)

Generalized Random Forests

The main competitor to double/debiased machine learning is **generalized**, or **causal**, random forests. The key papers are:

Susan Athey and Guido Imbens (2016), "Recursive partitioning for heterogeneous causal effects," *Proceedings of the National Academy of Sciences*, 113, 7353–7360. (2213 cites)

Stefan Wager and Susan Athey (2018), "Estimation and inference of heterogeneous treatment effects using random forests," *JASA*, 113, 1228–1242. (3681 cites)

Susan Athey, Julie Tibshirani, and Stefan Wager (2019), "Generalized random forests," *Annals of Statistics*, 47, 1148–1178. (2436 cites)

Generalized Random Forests

The main competitor to double/debiased machine learning is **generalized**, or **causal**, random forests. The key papers are:

Susan Athey and Guido Imbens (2016), "Recursive partitioning for heterogeneous causal effects," *Proceedings of the National Academy of Sciences*, 113, 7353–7360. (2213 cites)

Stefan Wager and Susan Athey (2018), "Estimation and inference of heterogeneous treatment effects using random forests," *JASA*, 113, 1228–1242. (3681 cites)

Susan Athey, Julie Tibshirani, and Stefan Wager (2019), "Generalized random forests," *Annals of Statistics*, 47, 1148–1178. (2436 cites)

There is an R package called `grf` which, like the `DoubleML` package, has a great many capabilities.

Generalized Random Forests

The main competitor to double/debiased machine learning is **generalized**, or **causal**, random forests. The key papers are:

Susan Athey and Guido Imbens (2016), "Recursive partitioning for heterogeneous causal effects," *Proceedings of the National Academy of Sciences*, 113, 7353–7360. (2213 cites)

Stefan Wager and Susan Athey (2018), "Estimation and inference of heterogeneous treatment effects using random forests," *JASA*, 113, 1228–1242. (3681 cites)

Susan Athey, Julie Tibshirani, and Stefan Wager (2019), "Generalized random forests," *Annals of Statistics*, 47, 1148–1178. (2436 cites)

There is an R package called `grf` which, like the `DoubleML` package, has a great many capabilities.

The R package uses C++ for efficiency, and the C++ code is available.

Causal Trees

Causal Trees

Athey and Imbens (2016) proposed to construct trees based on heterogeneity of treatment effects rather than prediction errors.

Causal Trees

Athey and Imbens (2016) proposed to construct trees based on heterogeneity of treatment effects rather than prediction errors.

They also proposed to construct **honest trees**. The sample is divided into two parts.

Causal Trees

Athey and Imbens (2016) proposed to construct trees based on heterogeneity of treatment effects rather than prediction errors.

They also proposed to construct **honest trees**. The sample is divided into two parts.

- One part of the training sample is used to construct the trees, that is, to decide where each of the splits is to occur.

Causal Trees

Athey and Imbens (2016) proposed to construct trees based on heterogeneity of treatment effects rather than prediction errors.

They also proposed to construct **honest trees**. The sample is divided into two parts.

- One part of the training sample is used to construct the trees, that is, to decide where each of the splits is to occur.
- If there is cross-validation, it also uses this part of the sample.

Causal Trees

Athey and Imbens (2016) proposed to construct trees based on heterogeneity of treatment effects rather than prediction errors.

They also proposed to construct **honest trees**. The sample is divided into two parts.

- One part of the training sample is used to construct the trees, that is, to decide where each of the splits is to occur.
- If there is cross-validation, it also uses this part of the sample.
- However, cross-validation does not focus on the bias-variance tradeoff, because the trees are honest. Instead, it focuses on the sizes of the leaves.

Causal Trees

Athey and Imbens (2016) proposed to construct trees based on heterogeneity of treatment effects rather than prediction errors.

They also proposed to construct **honest trees**. The sample is divided into two parts.

- One part of the training sample is used to construct the trees, that is, to decide where each of the splits is to occur.
- If there is cross-validation, it also uses this part of the sample.
- However, cross-validation does not focus on the bias-variance tradeoff, because the trees are honest. Instead, it focuses on the sizes of the leaves.
- Smaller leaves potentially lead to better predictions, but estimation error is larger because they have fewer observations.

Causal Trees

Athey and Imbens (2016) proposed to construct trees based on heterogeneity of treatment effects rather than prediction errors.

They also proposed to construct **honest trees**. The sample is divided into two parts.

- One part of the training sample is used to construct the trees, that is, to decide where each of the splits is to occur.
- If there is cross-validation, it also uses this part of the sample.
- However, cross-validation does not focus on the bias-variance tradeoff, because the trees are honest. Instead, it focuses on the sizes of the leaves.
- Smaller leaves potentially lead to better predictions, but estimation error is larger because they have fewer observations.
- The other part of the training sample is used to populate the leaves. Randomness in leaf averages should be independent of randomness in locations of the splits.

Honest estimation can be used when estimating both conditional means and **conditional average treatment effects**, or **CATE**.

Honest estimation can be used when estimating both conditional means and **conditional average treatment effects**, or **CATE**.

- Honest estimation is closely related to cross-fitting.

Honest estimation can be used when estimating both conditional means and **conditional average treatment effects**, or **CATE**.

- Honest estimation is closely related to cross-fitting.
- Both involve sample splitting and using part of the sample for one purpose and part of it for another purpose.

Honest estimation can be used when estimating both conditional means and **conditional average treatment effects**, or **CATE**.

- Honest estimation is closely related to cross-fitting.
- Both involve sample splitting and using part of the sample for one purpose and part of it for another purpose.
- A & I call the two parts the training and estimation samples.

Honest estimation can be used when estimating both conditional means and **conditional average treatment effects**, or **CATE**.

- Honest estimation is closely related to cross-fitting.
- Both involve sample splitting and using part of the sample for one purpose and part of it for another purpose.
- A & I call the two parts the training and estimation samples.
- The procedures for splitting and cross-validation are not the same as for conventional classification and regression trees.

Honest estimation can be used when estimating both conditional means and **conditional average treatment effects**, or **CATE**.

- Honest estimation is closely related to cross-fitting.
- Both involve sample splitting and using part of the sample for one purpose and part of it for another purpose.
- A & I call the two parts the training and estimation samples.
- The procedures for splitting and cross-validation are not the same as for conventional classification and regression trees.
- They take account of the fact that the leaves will be populated using an independent sample.

Honest estimation can be used when estimating both conditional means and **conditional average treatment effects**, or **CATE**.

- Honest estimation is closely related to cross-fitting.
- Both involve sample splitting and using part of the sample for one purpose and part of it for another purpose.
- A & I call the two parts the training and estimation samples.
- The procedures for splitting and cross-validation are not the same as for conventional classification and regression trees.
- They take account of the fact that the leaves will be populated using an independent sample.

Recall that, for trees, cross-validation is used for pruning. But, because the trees are honest, making too many splits does not cause over-fitting.

Honest estimation can be used when estimating both conditional means and **conditional average treatment effects**, or **CATE**.

- Honest estimation is closely related to cross-fitting.
- Both involve sample splitting and using part of the sample for one purpose and part of it for another purpose.
- A & I call the two parts the training and estimation samples.
- The procedures for splitting and cross-validation are not the same as for conventional classification and regression trees.
- They take account of the fact that the leaves will be populated using an independent sample.

Recall that, for trees, cross-validation is used for pruning. But, because the trees are honest, making too many splits does not cause over-fitting.

With honest trees, it is generally not necessary to prune as aggressively as it would be for conventional (adaptive) trees.

When estimating treatment effects, we have to modify the objective function, because we do not directly observe treatment effects.

When estimating treatment effects, we have to modify the objective function, because we do not directly observe treatment effects.

- Every unit either is or is not treated. Thus, for any unit, we can only observe either $y_i(1)$ or $y_i(0)$.

When estimating treatment effects, we have to modify the objective function, because we do not directly observe treatment effects.

- Every unit either is or is not treated. Thus, for any unit, we can only observe either $y_i(1)$ or $y_i(0)$.
- We cannot observe $y_i(1) - y_i(0)$, which is the quantity for which we want to estimate the (conditional) mean.

When estimating treatment effects, we have to modify the objective function, because we do not directly observe treatment effects.

- Every unit either is or is not treated. Thus, for any unit, we can only observe either $y_i(1)$ or $y_i(0)$.
- We cannot observe $y_i(1) - y_i(0)$, which is the quantity for which we want to estimate the (conditional) mean.

A & I propose methods for solving this problem, for both conventional CART and honest CART.

When estimating treatment effects, we have to modify the objective function, because we do not directly observe treatment effects.

- Every unit either is or is not treated. Thus, for any unit, we can only observe either $y_i(1)$ or $y_i(0)$.
- We cannot observe $y_i(1) - y_i(0)$, which is the quantity for which we want to estimate the (conditional) mean.

A & I propose methods for solving this problem, for both conventional CART and honest CART.

These involve using different objective functions to decide how to make the splits and how to perform cross-validation.

When estimating treatment effects, we have to modify the objective function, because we do not directly observe treatment effects.

- Every unit either is or is not treated. Thus, for any unit, we can only observe either $y_i(1)$ or $y_i(0)$.
- We cannot observe $y_i(1) - y_i(0)$, which is the quantity for which we want to estimate the (conditional) mean.

A & I propose methods for solving this problem, for both conventional CART and honest CART.

These involve using different objective functions to decide how to make the splits and how to perform cross-validation.

The expressions for these functions are not super complicated, but they involve a lot of notation. There are two terms.

When estimating treatment effects, we have to modify the objective function, because we do not directly observe treatment effects.

- Every unit either is or is not treated. Thus, for any unit, we can only observe either $y_i(1)$ or $y_i(0)$.
- We cannot observe $y_i(1) - y_i(0)$, which is the quantity for which we want to estimate the (conditional) mean.

A & I propose methods for solving this problem, for both conventional CART and honest CART.

These involve using different objective functions to decide how to make the splits and how to perform cross-validation.

The expressions for these functions are not super complicated, but they involve a lot of notation. There are two terms.

A partition is rewarded for finding strong heterogeneity in treatment effects and penalized for creating too much variance in leaf estimates.

In the prediction case, the two terms in the objective function both tend to select features that predict heterogeneity in outcomes.

In the prediction case, the two terms in the objective function both tend to select features that predict heterogeneity in outcomes.

In the treatment effect case, however, the two terms reward different types of features.

In the prediction case, the two terms in the objective function both tend to select features that predict heterogeneity in outcomes.

In the treatment effect case, however, the two terms reward different types of features.

- The variance of a treatment effect estimator can be reduced by introducing a split, even if both child leaves have the same average treatment effect.

In the prediction case, the two terms in the objective function both tend to select features that predict heterogeneity in outcomes.

In the treatment effect case, however, the two terms reward different types of features.

- The variance of a treatment effect estimator can be reduced by introducing a split, even if both child leaves have the same average treatment effect.
- This can happen if a covariate affects the mean outcome but not the treatment effects.

In the prediction case, the two terms in the objective function both tend to select features that predict heterogeneity in outcomes.

In the treatment effect case, however, the two terms reward different types of features.

- The variance of a treatment effect estimator can be reduced by introducing a split, even if both child leaves have the same average treatment effect.
- This can happen if a covariate affects the mean outcome but not the treatment effects.
- Such a split results in more homogeneous leaves, and thus lower-variance estimates of the means of the treatment group and control group outcomes.

In the prediction case, the two terms in the objective function both tend to select features that predict heterogeneity in outcomes.

In the treatment effect case, however, the two terms reward different types of features.

- The variance of a treatment effect estimator can be reduced by introducing a split, even if both child leaves have the same average treatment effect.
- This can happen if a covariate affects the mean outcome but not the treatment effects.
- Such a split results in more homogeneous leaves, and thus lower-variance estimates of the means of the treatment group and control group outcomes.
- Thus the distinction between the adaptive and honest splitting criteria will be more pronounced for treatment effect estimation.

Causal Forests

Causal Forests

Wager and Athey (2018) extended causal trees to **causal forests**.

Causal Forests

Wager and Athey (2018) extended causal trees to **causal forests**.

Athey, Tibshirani, and Wager (2019) further extended causal forests to **generalized random forests**.

Causal Forests

Wager and Athey (2018) extended causal trees to **causal forests**.

Athey, Tibshirani, and Wager (2019) further extended causal forests to **generalized random forests**.

The `grf` package implements both causal and regression forests, using the `causal_forest` and `regression_forest` functions.

Causal Forests

Wager and Athey (2018) extended causal trees to **causal forests**.

Athey, Tibshirani, and Wager (2019) further extended causal forests to **generalized random forests**.

The `grf` package implements both causal and regression forests, using the `causal_forest` and `regression_forest` functions.

- Like causal trees, causal forests are trying to estimate treatment effects, which are allowed to be heterogeneous.

Causal Forests

Wager and Athey (2018) extended causal trees to **causal forests**.

Athey, Tibshirani, and Wager (2019) further extended causal forests to **generalized random forests**.

The `grf` package implements both causal and regression forests, using the `causal_forest` and `regression_forest` functions.

- Like causal trees, causal forests are trying to estimate treatment effects, which are allowed to be heterogeneous.
- The effect of d_i on y_i is allowed to vary with x_i, so the model is explicitly not a partial linear model.

Causal Forests

Wager and Athey (2018) extended causal trees to **causal forests**.

Athey, Tibshirani, and Wager (2019) further extended causal forests to **generalized random forests**.

The `grf` package implements both causal and regression forests, using the `causal_forest` and `regression_forest` functions.

- Like causal trees, causal forests are trying to estimate treatment effects, which are allowed to be heterogeneous.
- The effect of d_i on y_i is allowed to vary with x_i, so the model is explicitly not a partial linear model.
- The splits are designed to yield large differences in treatment effects across nodes.

Causal Forests

Wager and Athey (2018) extended causal trees to **causal forests**.

Athey, Tibshirani, and Wager (2019) further extended causal forests to **generalized random forests**.

The `grf` package implements both causal and regression forests, using the `causal_forest` and `regression_forest` functions.

- Like causal trees, causal forests are trying to estimate treatment effects, which are allowed to be heterogeneous.
- The effect of d_i on y_i is allowed to vary with $\boldsymbol{x}_i$, so the model is explicitly not a partial linear model.
- The splits are designed to yield large differences in treatment effects across nodes.

The way predictions are made for causal forests differs from the usual method for random forests.

In the usual method, every tree makes a prediction for each observation in the test sample, and these are either averaged (for regression or predicting probabilities) or converted into a classification by majority voting.

In the usual method, every tree makes a prediction for each observation in the test sample, and these are either averaged (for regression or predicting probabilities) or converted into a classification by majority voting.

With GRF, however, we first figure out which leaf every test observation falls into.

In the usual method, every tree makes a prediction for each observation in the test sample, and these are either averaged (for regression or predicting probabilities) or converted into a classification by majority voting.

With GRF, however, we first figure out which leaf every test observation falls into.

Then we create a list of neighboring training observations, weighted by how many times each of them fell into the same leaf.

In the usual method, every tree makes a prediction for each observation in the test sample, and these are either averaged (for regression or predicting probabilities) or converted into a classification by majority voting.

With GRF, however, we first figure out which leaf every test observation falls into.

Then we create a list of neighboring training observations, weighted by how many times each of them fell into the same leaf.

- For regression forests, the prediction is the weighted average of the outcomes for the neighbors, really just the usual method.

In the usual method, every tree makes a prediction for each observation in the test sample, and these are either averaged (for regression or predicting probabilities) or converted into a classification by majority voting.

With GRF, however, we first figure out which leaf every test observation falls into.

Then we create a list of neighboring training observations, weighted by how many times each of them fell into the same leaf.

- For regression forests, the prediction is the weighted average of the outcomes for the neighbors, really just the usual method.
- For causal forests, the prediction is the treatment effect based on the weighted outcomes and treatment status of the neighbors.

In the usual method, every tree makes a prediction for each observation in the test sample, and these are either averaged (for regression or predicting probabilities) or converted into a classification by majority voting.

With GRF, however, we first figure out which leaf every test observation falls into.

Then we create a list of neighboring training observations, weighted by how many times each of them fell into the same leaf.

- For regression forests, the prediction is the weighted average of the outcomes for the neighbors, really just the usual method.
- For causal forests, the prediction is the treatment effect based on the weighted outcomes and treatment status of the neighbors.
- Thus GRF uses random forests as an adaptive nearest neighbor method; see (1).

In the usual method, every tree makes a prediction for each observation in the test sample, and these are either averaged (for regression or predicting probabilities) or converted into a classification by majority voting.

With GRF, however, we first figure out which leaf every test observation falls into.

Then we create a list of neighboring training observations, weighted by how many times each of them fell into the same leaf.

- For regression forests, the prediction is the weighted average of the outcomes for the neighbors, really just the usual method.
- For causal forests, the prediction is the treatment effect based on the weighted outcomes and treatment status of the neighbors.
- Thus GRF uses random forests as an adaptive nearest neighbor method; see (1).
- The algorithm finds a weighted set of neighbors that are similar to a test point, where there is more than one notion of similarity.

Figure 22.1 — The random forest weighting function

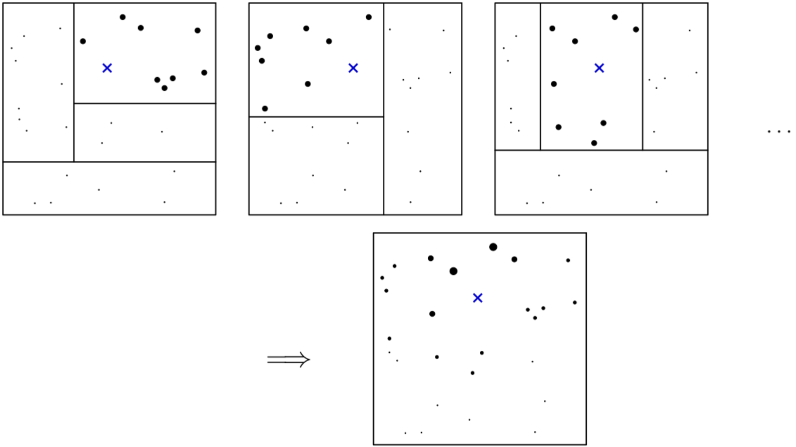


FIG. 1. *Illustration of the random forest weighting function. Each tree starts by giving equal (positive) weight to the training examples in the same leaf as our test point x of interest, and zero weight to all the other training examples. Then the forest averages all these tree-based weightings, and effectively measures how often each training example falls into the same leaf as x.*

The `causal_forest` and `regression_forest` functions perform the splits in different ways.

The `causal_forest` and `regression_forest` functions perform the splits in different ways.

- For regression, splitting is based directly on the outcomes, as in traditional regression tree algorithms.

The `causal_forest` and `regression_forest` functions perform the splits in different ways.

- For regression, splitting is based directly on the outcomes, as in traditional regression tree algorithms.
- For causal (CATE) estimation, it is based on how dissimilar the estimated treatment effects are.

The `causal_forest` and `regression_forest` functions perform the splits in different ways.

- For regression, splitting is based directly on the outcomes, as in traditional regression tree algorithms.
- For causal (CATE) estimation, it is based on how dissimilar the estimated treatment effects are.
- When a split is made, the algorithm wants the treatment effects to differ as much as possible.

The `causal_forest` and `regression_forest` functions perform the splits in different ways.

- For regression, splitting is based directly on the outcomes, as in traditional regression tree algorithms.
- For causal (CATE) estimation, it is based on how dissimilar the estimated treatment effects are.
- When a split is made, the algorithm wants the treatment effects to differ as much as possible.
- It does not make sense to ask for individual tree predictions. It is not individual trees that make predictions, but rather the neighborhood they induce.

The `causal_forest` and `regression_forest` functions perform the splits in different ways.

- For regression, splitting is based directly on the outcomes, as in traditional regression tree algorithms.
- For causal (CATE) estimation, it is based on how dissimilar the estimated treatment effects are.
- When a split is made, the algorithm wants the treatment effects to differ as much as possible.
- It does not make sense to ask for individual tree predictions. It is not individual trees that make predictions, but rather the neighborhood they induce.

Instead of bootstrapping, `grf` uses **subsampling**, that is, resampling without replacement. Each subsample contains `sample.fraction` (default 0.5) of the original sample.

The `causal_forest` and `regression_forest` functions perform the splits in different ways.

- For regression, splitting is based directly on the outcomes, as in traditional regression tree algorithms.
- For causal (CATE) estimation, it is based on how dissimilar the estimated treatment effects are.
- When a split is made, the algorithm wants the treatment effects to differ as much as possible.
- It does not make sense to ask for individual tree predictions. It is not individual trees that make predictions, but rather the neighborhood they induce.

Instead of bootstrapping, `grf` uses **subsampling**, that is, resampling without replacement. Each subsample contains `sample.fraction` (default 0.5) of the original sample.

Observations not in the current subsample are still called OOB observations, but we get to choose the OOB fraction instead of getting a random number that averages 36.8%.

The `causal_forest` function estimates causal forests.

The `causal_forest` function estimates causal forests.

When choosing a split, the algorithm attempts to maximize the difference in treatment effect between the two child nodes.

The `causal_forest` function estimates causal forests.

When choosing a split, the algorithm attempts to maximize the difference in treatment effect between the two child nodes.

This is explained in Athey, Tibshirani, and Wager (2019). It is related to what Athey and Imbens (2016) do in their causal trees paper, but a lot more complicated.

The `causal_forest` function estimates causal forests.

When choosing a split, the algorithm attempts to maximize the difference in treatment effect between the two child nodes.

This is explained in Athey, Tibshirani, and Wager (2019). It is related to what Athey and Imbens (2016) do in their causal trees paper, but a lot more complicated.

They call it the **gradient tree algorithm**.

The `causal_forest` function estimates causal forests.

When choosing a split, the algorithm attempts to maximize the difference in treatment effect between the two child nodes.

This is explained in Athey, Tibshirani, and Wager (2019). It is related to what Athey and Imbens (2016) do in their causal trees paper, but a lot more complicated.

They call it the **gradient tree algorithm**.

The gradients of an objective function are used to create pseudo-outcomes that are then treated as if they were regression outcomes when the splitting takes place.

The `causal_forest` function estimates causal forests.

When choosing a split, the algorithm attempts to maximize the difference in treatment effect between the two child nodes.

This is explained in Athey, Tibshirani, and Wager (2019). It is related to what Athey and Imbens (2016) do in their causal trees paper, but a lot more complicated.

They call it the **gradient tree algorithm**.

The gradients of an objective function are used to create pseudo-outcomes that are then treated as if they were regression outcomes when the splitting takes place.

What is actually maximized is a linear approximation based on pre-computed gradients for all the observations.

The `causal_forest` function estimates causal forests.

When choosing a split, the algorithm attempts to maximize the difference in treatment effect between the two child nodes.

This is explained in Athey, Tibshirani, and Wager (2019). It is related to what Athey and Imbens (2016) do in their causal trees paper, but a lot more complicated.

They call it the **gradient tree algorithm**.

The gradients of an objective function are used to create pseudo-outcomes that are then treated as if they were regression outcomes when the splitting takes place.

What is actually maximized is a linear approximation based on pre-computed gradients for all the observations.

Part of the complexity arises from the fact that ATW's trees can handle general estimating equations, not just regressions.

Before GRF creates a causal forest, it is orthogonalized using what the package documentation calls “Robinson’s transformation” but is actually just forest-based double-ML partialling out.

Before GRF creates a causal forest, it is orthogonalized using what the package documentation calls "Robinson's transformation" but is actually just forest-based double-ML partialling out.

- GRF first computes estimates of the propensity scores $m(\boldsymbol{x}_i) = \mathrm{E}(d_i|\boldsymbol{x}_i)$ and the marginal outcomes $g_i(\boldsymbol{x}_i) = \mathrm{E}(y_i|\boldsymbol{x}_i)$.

Before GRF creates a causal forest, it is orthogonalized using what the package documentation calls "Robinson's transformation" but is actually just forest-based double-ML partialling out.

- GRF first computes estimates of the propensity scores $m(\boldsymbol{x}_i) = \mathrm{E}(d_i|\boldsymbol{x}_i)$ and the marginal outcomes $g_i(\boldsymbol{x}_i) = \mathrm{E}(y_i|\boldsymbol{x}_i)$.
- Here I am using the notation from DDML, not the notation used by GRF. They use W for d and $\mu(\cdot)$ for $g(\cdot)$.

Before GRF creates a causal forest, it is orthogonalized using what the package documentation calls "Robinson's transformation" but is actually just forest-based double-ML partialling out.

- GRF first computes estimates of the propensity scores $m(\boldsymbol{x}_i) = \mathrm{E}(d_i|\boldsymbol{x}_i)$ and the marginal outcomes $g_i(\boldsymbol{x}_i) = \mathrm{E}(y_i|\boldsymbol{x}_i)$.
- Here I am using the notation from DDML, not the notation used by GRF. They use W for d and $\mu(\cdot)$ for $g(\cdot)$.
- GRF computes out-of-bag residuals $\hat{v}_i = d_i - \hat{m}_i$ and $\hat{u}_i - y_i - \hat{g}_i$. Then it trains a causal random forest on these residuals.

Before GRF creates a causal forest, it is orthogonalized using what the package documentation calls "Robinson's transformation" but is actually just forest-based double-ML partialling out.

- GRF first computes estimates of the propensity scores $m(\boldsymbol{x}_i) = \mathrm{E}(d_i|\boldsymbol{x}_i)$ and the marginal outcomes $g_i(\boldsymbol{x}_i) = \mathrm{E}(y_i|\boldsymbol{x}_i)$.
- Here I am using the notation from DDML, not the notation used by GRF. They use W for d and $\mu(\cdot)$ for $g(\cdot)$.
- GRF computes out-of-bag residuals $\hat{v}_i = d_i - \hat{m}_i$ and $\hat{u}_i - y_i - \hat{g}_i$. Then it trains a causal random forest on these residuals.
- Users can use other methods to obtain $\hat{m}_i$, or `W.hat`, and $\hat{g}_i$, or `Y.hat` and then feed these into the `causal_forest` function.

Before GRF creates a causal forest, it is orthogonalized using what the package documentation calls "Robinson's transformation" but is actually just forest-based double-ML partialling out.

- GRF first computes estimates of the propensity scores $m(\boldsymbol{x}_i) = \mathrm{E}(d_i|\boldsymbol{x}_i)$ and the marginal outcomes $g_i(\boldsymbol{x}_i) = \mathrm{E}(y_i|\boldsymbol{x}_i)$.
- Here I am using the notation from DDML, not the notation used by GRF. They use W for d and $\mu(\cdot)$ for $g(\cdot)$.
- GRF computes out-of-bag residuals $\hat{v}_i = d_i - \hat{m}_i$ and $\hat{u}_i - y_i - \hat{g}_i$. Then it trains a causal random forest on these residuals.
- Users can use other methods to obtain $\hat{m}_i$, or `W.hat`, and $\hat{g}_i$, or `Y.hat` and then feed these into the `causal_forest` function.

The out-of-bag residuals are based on leave-one-out estimates, not the part of the sample omitted when growing the forest.

Before GRF creates a causal forest, it is orthogonalized using what the package documentation calls “Robinson’s transformation” but is actually just forest-based double-ML partialling out.

- GRF first computes estimates of the propensity scores $m(\boldsymbol{x}_i) = \mathrm{E}(d_i|\boldsymbol{x}_i)$ and the marginal outcomes $g_i(\boldsymbol{x}_i) = \mathrm{E}(y_i|\boldsymbol{x}_i)$.
- Here I am using the notation from DDML, not the notation used by GRF. They use W for d and $\mu(\cdot)$ for $g(\cdot)$.
- GRF computes out-of-bag residuals $\hat{v}_i = d_i - \hat{m}_i$ and $\hat{u}_i - y_i - \hat{g}_i$. Then it trains a causal random forest on these residuals.
- Users can use other methods to obtain $\hat{m}_i$, or `W.hat`, and $\hat{g}_i$, or `Y.hat` and then feed these into the `causal_forest` function.

The out-of-bag residuals are based on leave-one-out estimates, not the part of the sample omitted when growing the forest.

Thus each forest is based solely on a particular subsample.

This is very similar to the interactive regression model of the DoubleML package, but (by default) based on regression forests instead of several possible ML methods.

This is very similar to the interactive regression model of the DoubleML package, but (by default) based on regression forests instead of several possible ML methods.

The original paper calls this **local centering**. The package documents call it **R-learning** (where “R” stands for “Robinson”).

This is very similar to the interactive regression model of the DoubleML package, but (by default) based on regression forests instead of several possible ML methods.

The original paper calls this **local centering**. The package documents call it **R-learning** (where “R” stands for “Robinson”).

ATW (2019) cites Robinson (1988), Newey (1994), and Chernozhukov et al. (2018). It is really using Neyman orthogonal scores!

This is very similar to the interactive regression model of the DoubleML package, but (by default) based on regression forests instead of several possible ML methods.

The original paper calls this **local centering**. The package documents call it **R-learning** (where “R” stands for “Robinson”).

ATW (2019) cites Robinson (1988), Newey (1994), and Chernozhukov et al. (2018). It is really using Neyman orthogonal scores!

It is easy to modify a previously constructed tree by dropping one observation at a time, as there will often be no change to the splits. Hence this quote from ATW:

This is very similar to the interactive regression model of the DoubleML package, but (by default) based on regression forests instead of several possible ML methods.

The original paper calls this **local centering**. The package documents call it **R-learning** (where "R" stands for "Robinson").

ATW (2019) cites Robinson (1988), Newey (1994), and Chernozhukov et al. (2018). It is really using Neyman orthogonal scores!

It is easy to modify a previously constructed tree by dropping one observation at a time, as there will often be no change to the splits. Hence this quote from ATW:

In the context of forests, it is much more practical to carry out residualization via leave-one-out prediction than via K-fold cross-fitting, because leave-one-out prediction in forests is computationally cheap [Breiman (2001)]; however, a practitioner wanting to use results that are precisely covered by theory may prefer to use cross-fitting for centering.

Honesty

Honesty

In a classic random forest, each bootstrap sample is used to choose a tree's splits and also to populate the leaf nodes.

Honesty

In a classic random forest, each bootstrap sample is used to choose a tree's splits and also to populate the leaf nodes.

Honest forests randomly split the bootstrap sample (which in this case is a subsample) in half and use only the first half to perform splitting.

Honesty

In a classic random forest, each bootstrap sample is used to choose a tree's splits and also to populate the leaf nodes.

Honest forests randomly split the bootstrap sample (which in this case is a subsample) in half and use only the first half to perform splitting.

The second half, which should be completely independent, is then used to populate the leaf nodes.

Honesty

In a classic random forest, each bootstrap sample is used to choose a tree's splits and also to populate the leaf nodes.

Honest forests randomly split the bootstrap sample (which in this case is a subsample) in half and use only the first half to perform splitting.

The second half, which should be completely independent, is then used to populate the leaf nodes.

Honesty plays a role similar to cross-fitting, but the splitting happens before every tree is grown.

Honesty

In a classic random forest, each bootstrap sample is used to choose a tree's splits and also to populate the leaf nodes.

Honest forests randomly split the bootstrap sample (which in this case is a subsample) in half and use only the first half to perform splitting.

The second half, which should be completely independent, is then used to populate the leaf nodes.

Honesty plays a role similar to cross-fitting, but the splitting happens before every tree is grown.

The DDML folks implicitly admit that cross-fitting makes the results too dependent on the random choice of folds. That is why they suggest repeating the procedure a number of times.

Honesty

In a classic random forest, each bootstrap sample is used to choose a tree's splits and also to populate the leaf nodes.

Honest forests randomly split the bootstrap sample (which in this case is a subsample) in half and use only the first half to perform splitting.

The second half, which should be completely independent, is then used to populate the leaf nodes.

Honesty plays a role similar to cross-fitting, but the splitting happens before every tree is grown.

The DDML folks implicitly admit that cross-fitting makes the results too dependent on the random choice of folds. That is why they suggest repeating the procedure a number of times.

If the dataset is small, honesty can hurt. It can be disabled by setting `honesty=FALSE`.

A less radical approach is to set `honesty.fraction` to a number larger than the default of 0.5, up to 0.8 or so.

A less radical approach is to set `honesty.fraction` to a number larger than the default of 0.5, up to 0.8 or so.

But when `honesty.fraction` is large, there will be more empty leaves, because there are less data to populate them, so more trees are needed.

A less radical approach is to set `honesty.fraction` to a number larger than the default of 0.5, up to 0.8 or so.

But when `honesty.fraction` is large, there will be more empty leaves, because there are less data to populate them, so more trees are needed.

- By default, `honesty.prune.leaves = TRUE`. In this case, trees are pruned to avoid empty leaves.

A less radical approach is to set `honesty.fraction` to a number larger than the default of 0.5, up to 0.8 or so.

But when `honesty.fraction` is large, there will be more empty leaves, because there are less data to populate them, so more trees are needed.

- By default, `honesty.prune.leaves = TRUE`. In this case, trees are pruned to avoid empty leaves.
- For small samples, it may be better to set it to `FALSE`. If so, empty leaves are not pruned.

A less radical approach is to set `honesty.fraction` to a number larger than the default of 0.5, up to 0.8 or so.

But when `honesty.fraction` is large, there will be more empty leaves, because there are less data to populate them, so more trees are needed.

- By default, `honesty.prune.leaves = TRUE`. In this case, trees are pruned to avoid empty leaves.
- For small samples, it may be better to set it to `FALSE`. If so, empty leaves are not pruned.
- In that case, some trees cannot predict some observations, and those trees are dropped when computing estimates.

A less radical approach is to set `honesty.fraction` to a number larger than the default of 0.5, up to 0.8 or so.

But when `honesty.fraction` is large, there will be more empty leaves, because there are less data to populate them, so more trees are needed.

- By default, `honesty.prune.leaves = TRUE`. In this case, trees are pruned to avoid empty leaves.
- For small samples, it may be better to set it to `FALSE`. If so, empty leaves are not pruned.
- In that case, some trees cannot predict some observations, and those trees are dropped when computing estimates.

If we set `sample.fraction` to a number greater than 0.5, the two "honest" subsamples would be larger. But this may not be a good thing to do.

A less radical approach is to set `honesty.fraction` to a number larger than the default of 0.5, up to 0.8 or so.

But when `honesty.fraction` is large, there will be more empty leaves, because there are less data to populate them, so more trees are needed.

- By default, `honesty.prune.leaves = TRUE`. In this case, trees are pruned to avoid empty leaves.
- For small samples, it may be better to set it to `FALSE`. If so, empty leaves are not pruned.
- In that case, some trees cannot predict some observations, and those trees are dropped when computing estimates.

If we set `sample.fraction` to a number greater than 0.5, the two "honest" subsamples would be larger. But this may not be a good thing to do.

There are evidently lots of tuning parameters to play with!

Variance Estimation

Variance Estimation

One nice feature of GRF is that it provides a way to obtain variances.

Variance Estimation

One nice feature of GRF is that it provides a way to obtain variances.

This involves a subsampling procedure described in Section 4 of Athey, Tibshirani, and Wager (2019).

Variance Estimation

One nice feature of GRF is that it provides a way to obtain variances.

This involves a subsampling procedure described in Section 4 of Athey, Tibshirani, and Wager (2019).

- The procedures involves a "bootstrap of little bags."

Variance Estimation

One nice feature of GRF is that it provides a way to obtain variances.

This involves a subsampling procedure described in Section 4 of Athey, Tibshirani, and Wager (2019).

- The procedures involves a “bootstrap of little bags.”
- It involves the difference between within-group and between-group variance estimates.

Variance Estimation

One nice feature of GRF is that it provides a way to obtain variances.

This involves a subsampling procedure described in Section 4 of Athey, Tibshirani, and Wager (2019).

- The procedures involves a “bootstrap of little bags.”
- It involves the difference between within-group and between-group variance estimates.
- If B, the number of subsamples, is small, the difference could have the wrong sign. This is worrying.

Variance Estimation

One nice feature of GRF is that it provides a way to obtain variances.

This involves a subsampling procedure described in Section 4 of Athey, Tibshirani, and Wager (2019).

- The procedures involves a "bootstrap of little bags."
- It involves the difference between within-group and between-group variance estimates.
- If B, the number of subsamples, is small, the difference could have the wrong sign. This is worrying.
- This method seems complicated but is allegedly inexpensive.

Variance Estimation

One nice feature of GRF is that it provides a way to obtain variances.

This involves a subsampling procedure described in Section 4 of Athey, Tibshirani, and Wager (2019).

- The procedures involves a "bootstrap of little bags."
- It involves the difference between within-group and between-group variance estimates.
- If B, the number of subsamples, is small, the difference could have the wrong sign. This is worrying.
- This method seems complicated but is allegedly inexpensive.
- When variance estimates are requested, `sample.fraction` cannot exceed 0.5, because it applies to half-samples but is expressed in terms of the full sample.

Clustering

Clustering

GRF provides support for cluster-robust forests by accounting for clusters in the subsampling process.

Clustering

GRF provides support for cluster-robust forests by accounting for clusters in the subsampling process.

The forest must be trained with the relevant cluster information by specifying the clusters and, possibly, the `equalize.cluster.weights` parameter.

Clustering

GRF provides support for cluster-robust forests by accounting for clusters in the subsampling process.

The forest must be trained with the relevant cluster information by specifying the clusters and, possibly, the `equalize.cluster.weights` parameter.

- All subsequent calls to `predict` will take clusters into account, including when estimating the variance of forest predictions.

Clustering

GRF provides support for cluster-robust forests by accounting for clusters in the subsampling process.

The forest must be trained with the relevant cluster information by specifying the clusters and, possibly, the `equalize.cluster.weights` parameter.

- All subsequent calls to `predict` will take clusters into account, including when estimating the variance of forest predictions.
- When clustering is enabled, all subsampling procedures operate on clusters instead of observations.

Clustering

GRF provides support for cluster-robust forests by accounting for clusters in the subsampling process.

The forest must be trained with the relevant cluster information by specifying the clusters and, possibly, the `equalize.cluster.weights` parameter.

- All subsequent calls to `predict` will take clusters into account, including when estimating the variance of forest predictions.
- When clustering is enabled, all subsampling procedures operate on clusters instead of observations.

When `equalize.cluster.weights` is TRUE, every cluster gets the same weight. When it is FALSE, every observation does.

Clustering

GRF provides support for cluster-robust forests by accounting for clusters in the subsampling process.

The forest must be trained with the relevant cluster information by specifying the clusters and, possibly, the `equalize.cluster.weights` parameter.

- All subsequent calls to `predict` will take clusters into account, including when estimating the variance of forest predictions.
- When clustering is enabled, all subsampling procedures operate on clusters instead of observations.

When `equalize.cluster.weights` is TRUE, every cluster gets the same weight. When it is FALSE, every observation does.

If cluster sizes differ a lot, giving each cluster the same weight may lead to substantial efficiency losses.

To determine the observations used for performing splitting and populating the leaves, `samples_per_cluster` examples are drawn from the selected clusters.

To determine the observations used for performing splitting and populating the leaves, `samples_per_cluster` examples are drawn from the selected clusters.

By default, `samples_per_cluster` is all the observations in each cluster, but it can be a fixed number.

To determine the observations used for performing splitting and populating the leaves, `samples_per_cluster` examples are drawn from the selected clusters.

By default, `samples_per_cluster` is all the observations in each cluster, but it can be a fixed number.

If `equalize.cluster.weights` is TRUE, `samples_per_cluster` is equal to the size of the smallest cluster.

To determine the observations used for performing splitting and populating the leaves, `samples_per_cluster` examples are drawn from the selected clusters.

By default, `samples_per_cluster` is all the observations in each cluster, but it can be a fixed number.

If `equalize.cluster.weights` is TRUE, `samples_per_cluster` is equal to the size of the smallest cluster.

- If `estimate.variance` is enabled, sample half of the clusters. If we are not computing variance estimates, then keep the full sample instead.

To determine the observations used for performing splitting and populating the leaves, `samples_per_cluster` examples are drawn from the selected clusters.

By default, `samples_per_cluster` is all the observations in each cluster, but it can be a fixed number.

If `equalize.cluster.weights` is TRUE, `samples_per_cluster` is equal to the size of the smallest cluster.

- If `estimate.variance` is enabled, sample half of the clusters. If we are not computing variance estimates, then keep the full sample instead.
- Sample `sample.fraction` of the clusters selected. Each tree is now associated with a list of cluster IDs.

To determine the observations used for performing splitting and populating the leaves, `samples_per_cluster` examples are drawn from the selected clusters.

By default, `samples_per_cluster` is all the observations in each cluster, but it can be a fixed number.

If `equalize.cluster.weights` is TRUE, `samples_per_cluster` is equal to the size of the smallest cluster.

- If `estimate.variance` is enabled, sample half of the clusters. If we are not computing variance estimates, then keep the full sample instead.
- Sample `sample.fraction` of the clusters selected. Each tree is now associated with a list of cluster IDs.
- If honesty is enabled, split these cluster IDs into two halves.

To determine the observations used for performing splitting and populating the leaves, `samples_per_cluster` examples are drawn from the selected clusters.

By default, `samples_per_cluster` is all the observations in each cluster, but it can be a fixed number.

If `equalize.cluster.weights` is TRUE, `samples_per_cluster` is equal to the size of the smallest cluster.

- If `estimate.variance` is enabled, sample half of the clusters. If we are not computing variance estimates, then keep the full sample instead.
- Sample `sample.fraction` of the clusters selected. Each tree is now associated with a list of cluster IDs.
- If honesty is enabled, split these cluster IDs into two halves.
- Draw `samples_per_cluster` observations from each of the cluster IDs, and do the same when repopulating the leaves for honesty.

Suppose that $G = 40$ and we are in the usual case where there is no weighting and every observation in each cluster is used.

Suppose that $G = 40$ and we are in the usual case where there is no weighting and every observation in each cluster is used.

Then each subsample will consist of 20 randomly drawn clusters, and the honest subsubsamples will contain just 10 clusters.

Suppose that $G = 40$ and we are in the usual case where there is no weighting and every observation in each cluster is used.

Then each subsample will consist of 20 randomly drawn clusters, and the honest subsubsamples will contain just 10 clusters.

When cluster sizes vary a lot, the numbers of observations used for splitting and populating may also vary a lot.

Suppose that $G = 40$ and we are in the usual case where there is no weighting and every observation in each cluster is used.

Then each subsample will consist of 20 randomly drawn clusters, and the honest subsubsamples will contain just 10 clusters.

When cluster sizes vary a lot, the numbers of observations used for splitting and populating may also vary a lot.

This may lead to poorly estimated variances, and it may be attractive to weight clusters equally, even if this leads to loss of efficiency.

Suppose that $G = 40$ and we are in the usual case where there is no weighting and every observation in each cluster is used.

Then each subsample will consist of 20 randomly drawn clusters, and the honest subsubsamples will contain just 10 clusters.

When cluster sizes vary a lot, the numbers of observations used for splitting and populating may also vary a lot.

This may lead to poorly estimated variances, and it may be attractive to weight clusters equally, even if this leads to loss of efficiency.

Athey and Wager (2019) applies GRF to a (synthetic) sample of 10,391 students drawn from 76 schools, based on a real study. They weight schools equally.

Suppose that $G = 40$ and we are in the usual case where there is no weighting and every observation in each cluster is used.

Then each subsample will consist of 20 randomly drawn clusters, and the honest subsubsamples will contain just 10 clusters.

When cluster sizes vary a lot, the numbers of observations used for splitting and populating may also vary a lot.

This may lead to poorly estimated variances, and it may be attractive to weight clusters equally, even if this leads to loss of efficiency.

Athey and Wager (2019) applies GRF to a (synthetic) sample of 10,391 students drawn from 76 schools, based on a real study. They weight schools equally.

Among other things, they find that clustering changes their results a lot, but orthogonalization (double ML) has little effect.

Suppose that $G = 40$ and we are in the usual case where there is no weighting and every observation in each cluster is used.

Then each subsample will consist of 20 randomly drawn clusters, and the honest subsubsamples will contain just 10 clusters.

When cluster sizes vary a lot, the numbers of observations used for splitting and populating may also vary a lot.

This may lead to poorly estimated variances, and it may be attractive to weight clusters equally, even if this leads to loss of efficiency.

Athey and Wager (2019) applies GRF to a (synthetic) sample of 10,391 students drawn from 76 schools, based on a real study. They weight schools equally.

Among other things, they find that clustering changes their results a lot, but orthogonalization (double ML) has little effect.

They speculate that this is because the main predictors in both the $g(\cdot)$ and $m(\cdot)$ functions are the same.

Boosted Regression Forests

Boosted Regression Forests

The `boosted_regression_forest` method trains a series of regression (not causal) forests.

Boosted Regression Forests

The `boosted_regression_forest` method trains a series of regression (not causal) forests.

Each forest is trained on the OOB residuals from the previous step.

Boosted Regression Forests

The `boosted_regression_forest` method trains a series of regression (not causal) forests.

Each forest is trained on the OOB residuals from the previous step.

There are parameters to control the step selection procedure. By default, the number of boosting steps is chosen by cross-validation:

Boosted Regression Forests

The `boosted_regression_forest` method trains a series of regression (not causal) forests.

Each forest is trained on the OOB residuals from the previous step.

There are parameters to control the step selection procedure. By default, the number of boosting steps is chosen by cross-validation:

- First, a single regression forest is trained and added to the initial, or previously boosted, forest.

Boosted Regression Forests

The `boosted_regression_forest` method trains a series of regression (not causal) forests.

Each forest is trained on the OOB residuals from the previous step.

There are parameters to control the step selection procedure. By default, the number of boosting steps is chosen by cross-validation:

- First, a single regression forest is trained and added to the initial, or previously boosted, forest.
- Next, a small forest of size `boost.trees.tune` is trained on the OOB residuals.

Boosted Regression Forests

The `boosted_regression_forest` method trains a series of regression (not causal) forests.

Each forest is trained on the OOB residuals from the previous step.

There are parameters to control the step selection procedure. By default, the number of boosting steps is chosen by cross-validation:

- First, a single regression forest is trained and added to the initial, or previously boosted, forest.
- Next, a small forest of size `boost.trees.tune` is trained on the OOB residuals.
- If its estimated OOB error reduces the OOB error from the previous step by more than `boost.error.reduction`, then we prepare to take another step.

Boosted Regression Forests

The `boosted_regression_forest` method trains a series of regression (not causal) forests.

Each forest is trained on the OOB residuals from the previous step.

There are parameters to control the step selection procedure. By default, the number of boosting steps is chosen by cross-validation:

- First, a single regression forest is trained and added to the initial, or previously boosted, forest.
- Next, a small forest of size `boost.trees.tune` is trained on the OOB residuals.
- If its estimated OOB error reduces the OOB error from the previous step by more than `boost.error.reduction`, then we prepare to take another step.
- In the next step, we train a full-sized regression forest on the residuals and add it to the boosted forest.

- This process continues until `boost.error.reduction` cannot be met when training the small forest, or if the total number of steps exceeds the limit `boost.max.steps`.

- This process continues until `boost.error.reduction` cannot be met when training the small forest, or if the total number of steps exceeds the limit `boost.max.steps`.

Alternatively, the cross-validation procedure can be skipped and the number of steps specified by the parameter `boost.steps`.

- This process continues until `boost.error.reduction` cannot be met when training the small forest, or if the total number of steps exceeds the limit `boost.max.steps`.

Alternatively, the cross-validation procedure can be skipped and the number of steps specified by the parameter `boost.steps`.

If `tune.parameters` is enabled, the parameters are chosen by the `regression_forest` procedure once in the first boosting step.

- This process continues until `boost.error.reduction` cannot be met when training the small forest, or if the total number of steps exceeds the limit `boost.max.steps`.

Alternatively, the cross-validation procedure can be skipped and the number of steps specified by the parameter `boost.steps`.

If `tune.parameters` is enabled, the parameters are chosen by the `regression_forest` procedure once in the first boosting step.

The selected parameters are then applied to train the forests in any further steps.

- This process continues until `boost.error.reduction` cannot be met when training the small forest, or if the total number of steps exceeds the limit `boost.max.steps`.

Alternatively, the cross-validation procedure can be skipped and the number of steps specified by the parameter `boost.steps`.

If `tune.parameters` is enabled, the parameters are chosen by the `regression_forest` procedure once in the first boosting step.

The selected parameters are then applied to train the forests in any further steps.

The `estimate.variance` parameter is not available for boosted forests.

- This process continues until `boost.error.reduction` cannot be met when training the small forest, or if the total number of steps exceeds the limit `boost.max.steps`.

Alternatively, the cross-validation procedure can be skipped and the number of steps specified by the parameter `boost.steps`.

If `tune.parameters` is enabled, the parameters are chosen by the `regression_forest` procedure once in the first boosting step.

The selected parameters are then applied to train the forests in any further steps.

The `estimate.variance` parameter is not available for boosted forests.

OOB predictions are available for the training data. These combine the OOB predictions for the forests from each boosting step.

- This process continues until `boost.error.reduction` cannot be met when training the small forest, or if the total number of steps exceeds the limit `boost.max.steps`.

Alternatively, the cross-validation procedure can be skipped and the number of steps specified by the parameter `boost.steps`.

If `tune.parameters` is enabled, the parameters are chosen by the `regression_forest` procedure once in the first boosting step.

The selected parameters are then applied to train the forests in any further steps.

The `estimate.variance` parameter is not available for boosted forests.

OOB predictions are available for the training data. These combine the OOB predictions for the forests from each boosting step.

Boosting seems to improve out-of-bag forest predictions most in scenarios where there is a strong signal-to-noise ratio.

Missing Values

Missing Values

GRF can handle missing covariate values. The input covariate matrix can contain `NA` in some cells instead of numerical values.

Missing Values

GRF can handle missing covariate values. The input covariate matrix can contain `NA` in some cells instead of numerical values.

These missing values can provide useful information, so the observations that contain them are not omitted.

Missing Values

GRF can handle missing covariate values. The input covariate matrix can contain `NA` in some cells instead of numerical values.

These missing values can provide useful information, so the observations that contain them are not omitted.

High incomes may be more likely to be missing than lower incomes.

Missing Values

GRF can handle missing covariate values. The input covariate matrix can contain `NA` in some cells instead of numerical values.

These missing values can provide useful information, so the observations that contain them are not omitted.

High incomes may be more likely to be missing than lower incomes.

As soon as there is a missing value in a variable j, there are three candidate splits for a given threshold:

Missing Values

GRF can handle missing covariate values. The input covariate matrix can contain `NA` in some cells instead of numerical values.

These missing values can provide useful information, so the observations that contain them are not omitted.

High incomes may be more likely to be missing than lower incomes.

As soon as there is a missing value in a variable j, there are three candidate splits for a given threshold:

- Split the samples with observed value for j according to the threshold and send all samples with missing value to the left;

Missing Values

GRF can handle missing covariate values. The input covariate matrix can contain `NA` in some cells instead of numerical values.

These missing values can provide useful information, so the observations that contain them are not omitted.

High incomes may be more likely to be missing than lower incomes.

As soon as there is a missing value in a variable j, there are three candidate splits for a given threshold:

- Split the samples with observed value for j according to the threshold and send all samples with missing value to the left;
- split the samples with observed value for j according to the threshold and send all samples with missing value to the right;

Missing Values

GRF can handle missing covariate values. The input covariate matrix can contain `NA` in some cells instead of numerical values.

These missing values can provide useful information, so the observations that contain them are not omitted.

High incomes may be more likely to be missing than lower incomes.

As soon as there is a missing value in a variable j, there are three candidate splits for a given threshold:

- Split the samples with observed value for j according to the threshold and send all samples with missing value to the left;
- split the samples with observed value for j according to the threshold and send all samples with missing value to the right;
- send all samples with observed value for j to the left and all samples with missing value for j to the right.

By default, missing values are sent to the left.

By default, missing values are sent to the left.

Therefore, if the covariates from the training set are completely observed but there are missing values in the test set, then the prediction functions will not produce an error.

By default, missing values are sent to the left.

Therefore, if the covariates from the training set are completely observed but there are missing values in the test set, then the prediction functions will not produce an error.

Instead, they simply send the incomplete observations to the left side of the corresponding nodes.

By default, missing values are sent to the left.

Therefore, if the covariates from the training set are completely observed but there are missing values in the test set, then the prediction functions will not produce an error.

Instead, they simply send the incomplete observations to the left side of the corresponding nodes.

This is something that any tree-based method could do, but precisely what each algorithm does varies.

By default, missing values are sent to the left.

Therefore, if the covariates from the training set are completely observed but there are missing values in the test set, then the prediction functions will not produce an error.

Instead, they simply send the incomplete observations to the left side of the corresponding nodes.

This is something that any tree-based method could do, but precisely what each algorithm does varies.

One possibility is to use methods based on imputation. The `randomForest` package can do this, using a method that seems a bit complicated.

By default, missing values are sent to the left.

Therefore, if the covariates from the training set are completely observed but there are missing values in the test set, then the prediction functions will not produce an error.

Instead, they simply send the incomplete observations to the left side of the corresponding nodes.

This is something that any tree-based method could do, but precisely what each algorithm does varies.

One possibility is to use methods based on imputation. The `randomForest` package can do this, using a method that seems a bit complicated.

These methods use relationships among explanatory variables to predict missing values, then replace `NA` with the predictions.

Local Linear Forests

Local Linear Forests

The `ll_regression_forest` command estimates the conditional mean function using a local[ly] linear random forest.

Local Linear Forests

The `ll_regression_forest` command estimates the conditional mean function using a local[ly] linear random forest.

The method was proposed in Friedberg, Tibshirani, Athey, and Wager (*Journal of Computational and Graphical Statistics*, 2020, 322 cites).

Local Linear Forests

The `ll_regression_forest` command estimates the conditional mean function using a local[ly] linear random forest.

The method was proposed in Friedberg, Tibshirani, Athey, and Wager (*Journal of Computational and Graphical Statistics*, 2020, 322 cites).

Recall that locally linear kernel regression often works much better than locally constant kernel regression.

Local Linear Forests

The `ll_regression_forest` command estimates the conditional mean function using a local[ly] linear random forest.

The method was proposed in Friedberg, Tibshirani, Athey, and Wager (*Journal of Computational and Graphical Statistics*, 2020, 322 cites).

Recall that locally linear kernel regression often works much better than locally constant kernel regression.

The same basic idea can be applied to random forests.

Local Linear Forests

The `ll_regression_forest` command estimates the conditional mean function using a local[ly] linear random forest.

The method was proposed in Friedberg, Tibshirani, Athey, and Wager (*Journal of Computational and Graphical Statistics*, 2020, 322 cites).

Recall that locally linear kernel regression often works much better than locally constant kernel regression.

The same basic idea can be applied to random forests.

For kernel regression, we use kernel weights, which may or may not go to zero for points far enough from x_0.

Local Linear Forests

The `ll_regression_forest` command estimates the conditional mean function using a local[ly] linear random forest.

The method was proposed in Friedberg, Tibshirani, Athey, and Wager (*Journal of Computational and Graphical Statistics*, 2020, 322 cites).

Recall that locally linear kernel regression often works much better than locally constant kernel regression.

The same basic idea can be applied to random forests.

For kernel regression, we use kernel weights, which may or may not go to zero for points far enough from x_0.

For LL regression forests, we use forest weights, just like the ones already used by GRF.

Local Linear Forests

The `ll_regression_forest` command estimates the conditional mean function using a local[ly] linear random forest.

The method was proposed in Friedberg, Tibshirani, Athey, and Wager (*Journal of Computational and Graphical Statistics*, 2020, 322 cites).

Recall that locally linear kernel regression often works much better than locally constant kernel regression.

The same basic idea can be applied to random forests.

For kernel regression, we use kernel weights, which may or may not go to zero for points far enough from x_0.

For LL regression forests, we use forest weights, just like the ones already used by GRF.

The weights for x_0 are given by the following equation:

$$\alpha_i(\boldsymbol{x}_0) = \frac{1}{B} \sum_{b=1}^{B} \frac{\mathbb{I}\left(\boldsymbol{x}_i \in L_b(\boldsymbol{x}_0)\right)}{|L_b(\boldsymbol{x}_0)|}. \tag{1}$$

$$\alpha_i(\boldsymbol{x}_0) = \frac{1}{B} \sum_{b=1}^{B} \frac{\mathbb{I}\left(\boldsymbol{x}_i \in L_b(\boldsymbol{x}_0)\right)}{|L_b(\boldsymbol{x}_0)|}. \tag{1}$$

Here $|L_b(\boldsymbol{x}_0)|$ is the number of points in the leaf containing $\boldsymbol{x}_0$ for subsample b.

$$\alpha_i(\boldsymbol{x}_0) = \frac{1}{B} \sum_{b=1}^{B} \frac{\mathbb{I}\left(\boldsymbol{x}_i \in L_b(\boldsymbol{x}_0)\right)}{|L_b(\boldsymbol{x}_0)|}. \tag{1}$$

Here $|L_b(\boldsymbol{x}_0)|$ is the number of points in the leaf containing $\boldsymbol{x}_0$ for subsample b.

This is just the weight you get by averaging the trees. From what ATW wrote, I thought they were using

$$\alpha_i'(\boldsymbol{x}_0) = \frac{\sum_{b=1}^{B} \mathbb{I}\left(\boldsymbol{x}_i \in L_b(\boldsymbol{x}_0)\right)}{\sum_{b=1}^{B} |L_b(\boldsymbol{x}_0)|}. \tag{2}$$

$$\alpha_i(\boldsymbol{x}_0) = \frac{1}{B}\sum_{b=1}^{B} \frac{\mathbb{I}\left(\boldsymbol{x}_i \in L_b(\boldsymbol{x}_0)\right)}{|L_b(\boldsymbol{x}_0)|}. \tag{1}$$

Here $|L_b(\boldsymbol{x}_0)|$ is the number of points in the leaf containing $\boldsymbol{x}_0$ for subsample b.

This is just the weight you get by averaging the trees. From what ATW wrote, I thought they were using

$$\alpha_i'(\boldsymbol{x}_0) = \frac{\sum_{b=1}^{B} \mathbb{I}\left(\boldsymbol{x}_i \in L_b(\boldsymbol{x}_0)\right)}{\sum_{b=1}^{B} |L_b(\boldsymbol{x}_0)|}. \tag{2}$$

When the number of points in the leaves to which $\boldsymbol{x}_0$ belongs vary across subsamples, (1) and (2) differ.

$$\alpha_i(\boldsymbol{x}_0) = \frac{1}{B} \sum_{b=1}^{B} \frac{\mathbb{I}\big(\boldsymbol{x}_i \in L_b(\boldsymbol{x}_0)\big)}{|L_b(\boldsymbol{x}_0)|}. \tag{1}$$

Here $|L_b(\boldsymbol{x}_0)|$ is the number of points in the leaf containing $\boldsymbol{x}_0$ for subsample b.

This is just the weight you get by averaging the trees. From what ATW wrote, I thought they were using

$$\alpha_i'(\boldsymbol{x}_0) = \frac{\sum_{b=1}^{B} \mathbb{I}\big(\boldsymbol{x}_i \in L_b(\boldsymbol{x}_0)\big)}{\sum_{b=1}^{B} |L_b(\boldsymbol{x}_0)|}. \tag{2}$$

When the number of points in the leaves to which $\boldsymbol{x}_0$ belongs vary across subsamples, (1) and (2) differ.

Normally, a random forest would take a weighted average, with weights $\alpha_i(\boldsymbol{x}_0)$, of the y_i in the leaves containing $\boldsymbol{x}_0$.

Instead, FTAW (2020) minimize

$$\sum_{i=1}^{n} \alpha_i(\boldsymbol{x}_0)\big(y_i - \mu(\boldsymbol{x}_0) - (\boldsymbol{x}_i - \boldsymbol{x}_0)\boldsymbol{\theta}(\boldsymbol{x}_0)\big)^2 + \lambda\|\boldsymbol{\theta}(\boldsymbol{x}_0)\|^2. \tag{3}$$

Instead, FTAW (2020) minimize

$$\sum_{i=1}^{n} \alpha_i(\boldsymbol{x}_0)\big(y_i - \mu(\boldsymbol{x}_0) - (\boldsymbol{x}_i - \boldsymbol{x}_0)\boldsymbol{\theta}(\boldsymbol{x}_0)\big)^2 + \lambda\|\boldsymbol{\theta}(\boldsymbol{x}_0)\|^2. \tag{3}$$

Here $\mu(\boldsymbol{x}_0)$ is an intercept, and $\boldsymbol{\theta}(\boldsymbol{x}_0)$ is a vector of slope coefficients.

Instead, FTAW (2020) minimize

$$\sum_{i=1}^{n} \alpha_i(\boldsymbol{x}_0)\big(y_i - \mu(\boldsymbol{x}_0) - (\boldsymbol{x}_i - \boldsymbol{x}_0)\boldsymbol{\theta}(\boldsymbol{x}_0)\big)^2 + \lambda\|\boldsymbol{\theta}(\boldsymbol{x}_0)\|^2. \tag{3}$$

Here $\mu(\boldsymbol{x}_0)$ is an intercept, and $\boldsymbol{\theta}(\boldsymbol{x}_0)$ is a vector of slope coefficients. The term $\lambda\|\boldsymbol{\theta}(\boldsymbol{x}_0)\|^2$ is a ridge penalty.

Instead, FTAW (2020) minimize

$$\sum_{i=1}^{n} \alpha_i(\boldsymbol{x}_0)\big(y_i - \mu(\boldsymbol{x}_0) - (\boldsymbol{x}_i - \boldsymbol{x}_0)\boldsymbol{\theta}(\boldsymbol{x}_0)\big)^2 + \lambda\|\boldsymbol{\theta}(\boldsymbol{x}_0)\|^2. \tag{3}$$

Here $\mu(\boldsymbol{x}_0)$ is an intercept, and $\boldsymbol{\theta}(\boldsymbol{x}_0)$ is a vector of slope coefficients.

The term $\lambda\|\boldsymbol{\theta}(\boldsymbol{x}_0)\|^2$ is a ridge penalty.

Categorical covariates, if any, are treated differently from the continuous ones. In many cases, they will not vary within leaves.

Instead, FTAW (2020) minimize

$$\sum_{i=1}^{n} \alpha_i(\boldsymbol{x}_0)\big(y_i - \mu(\boldsymbol{x}_0) - (\boldsymbol{x}_i - \boldsymbol{x}_0)\boldsymbol{\theta}(\boldsymbol{x}_0)\big)^2 + \lambda\|\boldsymbol{\theta}(\boldsymbol{x}_0)\|^2. \tag{3}$$

Here $\mu(\boldsymbol{x}_0)$ is an intercept, and $\boldsymbol{\theta}(\boldsymbol{x}_0)$ is a vector of slope coefficients.

The term $\lambda\|\boldsymbol{\theta}(\boldsymbol{x}_0)\|^2$ is a ridge penalty.

Categorical covariates, if any, are treated differently from the continuous ones. In many cases, they will not vary within leaves.

If we know that we will be using (3), the splitting procedure used to create the trees should be modified.

Instead, FTAW (2020) minimize

$$\sum_{i=1}^{n} \alpha_i(\boldsymbol{x}_0)\big(y_i - \mu(\boldsymbol{x}_0) - (\boldsymbol{x}_i - \boldsymbol{x}_0)\boldsymbol{\theta}(\boldsymbol{x}_0)\big)^2 + \lambda\|\boldsymbol{\theta}(\boldsymbol{x}_0)\|^2. \tag{3}$$

Here $\mu(\boldsymbol{x}_0)$ is an intercept, and $\boldsymbol{\theta}(\boldsymbol{x}_0)$ is a vector of slope coefficients.

The term $\lambda\|\boldsymbol{\theta}(\boldsymbol{x}_0)\|^2$ is a ridge penalty.

Categorical covariates, if any, are treated differently from the continuous ones. In many cases, they will not vary within leaves.

If we know that we will be using (3), the splitting procedure used to create the trees should be modified.

FTAW propose what they call **residual splitting**.

Instead, FTAW (2020) minimize

$$\sum_{i=1}^{n} \alpha_i(\boldsymbol{x}_0)\big(y_i - \mu(\boldsymbol{x}_0) - (\boldsymbol{x}_i - \boldsymbol{x}_0)\boldsymbol{\theta}(\boldsymbol{x}_0)\big)^2 + \lambda\|\boldsymbol{\theta}(\boldsymbol{x}_0)\|^2. \tag{3}$$

Here $\mu(\boldsymbol{x}_0)$ is an intercept, and $\boldsymbol{\theta}(\boldsymbol{x}_0)$ is a vector of slope coefficients.

The term $\lambda\|\boldsymbol{\theta}(\boldsymbol{x}_0)\|^2$ is a ridge penalty.

Categorical covariates, if any, are treated differently from the continuous ones. In many cases, they will not vary within leaves.

If we know that we will be using (3), the splitting procedure used to create the trees should be modified.

FTAW propose what they call **residual splitting**.

CART would simply choose the split that minimizes the SSR over the two means $\bar{y}_1$ and $\bar{y}_2$ conditional on $\boldsymbol{x}_i$ belonging to the child nodes C_1 and C_2 created by the split:

$$\mathrm{SSR}(C_1, C_2) = \sum_{\boldsymbol{x}_i \in C_1} (y_i - \bar{y}_1)^2 + \sum_{\boldsymbol{x}_i \in C_2} (y_i - \bar{y}_2)^2. \tag{4}$$

$$\mathrm{SSR}(C_1, C_2) = \sum_{\boldsymbol{x}_i \in C_1} (y_i - \bar{y}_1)^2 + \sum_{\boldsymbol{x}_i \in C_2} (y_i - \bar{y}_2)^2. \tag{4}$$

Instead, they first run a ridge regression within the parent node to predict y_i from $\boldsymbol{x}_i$.

$$\mathrm{SSR}(C_1, C_2) = \sum_{\boldsymbol{x}_i \in C_1} (y_i - \bar{y}_1)^2 + \sum_{\boldsymbol{x}_i \in C_2} (y_i - \bar{y}_2)^2. \tag{4}$$

Instead, they first run a ridge regression within the parent node to predict y_i from $\boldsymbol{x}_i$.

It yields fits $\hat{y}_i$ and residuals $\hat{u}_i = y_i - \hat{y}_i$.

$$\text{SSR}(C_1, C_2) = \sum_{\boldsymbol{x}_i \in C_1} (y_i - \bar{y}_1)^2 + \sum_{\boldsymbol{x}_i \in C_2} (y_i - \bar{y}_2)^2. \tag{4}$$

Instead, they first run a ridge regression within the parent node to predict y_i from $\boldsymbol{x}_i$.

It yields fits $\hat{y}_i$ and residuals $\hat{u}_i = y_i - \hat{y}_i$.

Then they choose the split that minimizes

$$\text{SSR}(C_1, C_2) = \sum_{\boldsymbol{x}_i \in C_1} (\hat{u}_i - \bar{u}_1)^2 + \sum_{\boldsymbol{x}_i \in C_2} (\hat{u}_i - \bar{u}_2)^2, \tag{5}$$

where $\bar{u}_1$ and $\bar{u}_2$ are the means of the residuals within each of the child nodes.

$$\mathrm{SSR}(C_1, C_2) = \sum_{\boldsymbol{x}_i \in C_1} (y_i - \bar{y}_1)^2 + \sum_{\boldsymbol{x}_i \in C_2} (y_i - \bar{y}_2)^2. \tag{4}$$

Instead, they first run a ridge regression within the parent node to predict y_i from $\boldsymbol{x}_i$.

It yields fits $\hat{y}_i$ and residuals $\hat{u}_i = y_i - \hat{y}_i$.

Then they choose the split that minimizes

$$\mathrm{SSR}(C_1, C_2) = \sum_{\boldsymbol{x}_i \in C_1} (\hat{u}_i - \bar{u}_1)^2 + \sum_{\boldsymbol{x}_i \in C_2} (\hat{u}_i - \bar{u}_2)^2, \tag{5}$$

where $\bar{u}_1$ and $\bar{u}_2$ are the means of the residuals within each of the child nodes.

Since smooth relationships will later be modeled by the local regressions, the tree-growing process focuses on what the local regressions cannot model well.

Figure 22.2 is taken from FTAW (2020).

Figure 22.2 is taken from FTAW (2020).

It uses fake data generated by the model

$$y_i = 20(x_{i1} - 0.5)^3 + \sum_{j=2}^{3} 10x_{ij} + \sum_{j=4}^{5} 5x_{ij} + \sum_{j=6}^{20} 2x_{ij}, \tag{6}$$

which apparently has no disturbance term (but it does).

Figure 22.2 is taken from FTAW (2020).

It uses fake data generated by the model

$$y_i = 20(x_{i1} - 0.5)^3 + \sum_{j=2}^{3} 10x_{ij} + \sum_{j=4}^{5} 5x_{ij} + \sum_{j=6}^{20} 2x_{ij}, \tag{6}$$

which apparently has no disturbance term (but it does).

As (6) is written, j runs from 1 to 20. However, the experiments actually use just d regressors, all $\mathrm{U}(0,1)$, where $d = 1, \ldots, 20$.

Figure 22.2 is taken from FTAW (2020).

It uses fake data generated by the model

$$y_i = 20(x_{i1} - 0.5)^3 + \sum_{j=2}^{3} 10x_{ij} + \sum_{j=4}^{5} 5x_{ij} + \sum_{j=6}^{20} 2x_{ij}, \qquad (6)$$

which apparently has no disturbance term (but it does).

As (6) is written, j runs from 1 to 20. However, the experiments actually use just d regressors, all $\mathrm{U}(0, 1)$, where $d = 1, \ldots, 20$.

When $d = 1$, there is just a cubic term, which is impossible to model by a linear regression.

Figure 22.2 is taken from FTAW (2020).

It uses fake data generated by the model

$$y_i = 20(x_{i1} - 0.5)^3 + \sum_{j=2}^{3} 10x_{ij} + \sum_{j=4}^{5} 5x_{ij} + \sum_{j=6}^{20} 2x_{ij}, \tag{6}$$

which apparently has no disturbance term (but it does).

As (6) is written, j runs from 1 to 20. However, the experiments actually use just d regressors, all $\mathrm{U}(0,1)$, where $d = 1, \ldots, 20$.

When $d = 1$, there is just a cubic term, which is impossible to model by a linear regression.

For $d = 2$ and $d = 3$, there are also strong linear terms.

Figure 22.2 is taken from FTAW (2020).

It uses fake data generated by the model

$$y_i = 20(x_{i1} - 0.5)^3 + \sum_{j=2}^{3} 10x_{ij} + \sum_{j=4}^{5} 5x_{ij} + \sum_{j=6}^{20} 2x_{ij}, \tag{6}$$

which apparently has no disturbance term (but it does).

As (6) is written, j runs from 1 to 20. However, the experiments actually use just d regressors, all $\mathrm{U}(0,1)$, where $d = 1, \ldots, 20$.

When $d = 1$, there is just a cubic term, which is impossible to model by a linear regression.

For $d = 2$ and $d = 3$, there are also strong linear terms.

For $d = 4$ and $d = 5$, there are fairly strong linear terms.

Figure 22.2 is taken from FTAW (2020).

It uses fake data generated by the model

$$y_i = 20(x_{i1} - 0.5)^3 + \sum_{j=2}^{3} 10x_{ij} + \sum_{j=4}^{5} 5x_{ij} + \sum_{j=6}^{20} 2x_{ij}, \tag{6}$$

which apparently has no disturbance term (but it does).

As (6) is written, j runs from 1 to 20. However, the experiments actually use just d regressors, all $\mathrm{U}(0,1)$, where $d = 1, \ldots, 20$.

When $d = 1$, there is just a cubic term, which is impossible to model by a linear regression.

For $d = 2$ and $d = 3$, there are also strong linear terms.

For $d = 4$ and $d = 5$, there are fairly strong linear terms.

For larger values of d, there are even more linear terms, but the last 15 of them are quite weak.

Figure 22.2 — Random forest versus locally linear random forest

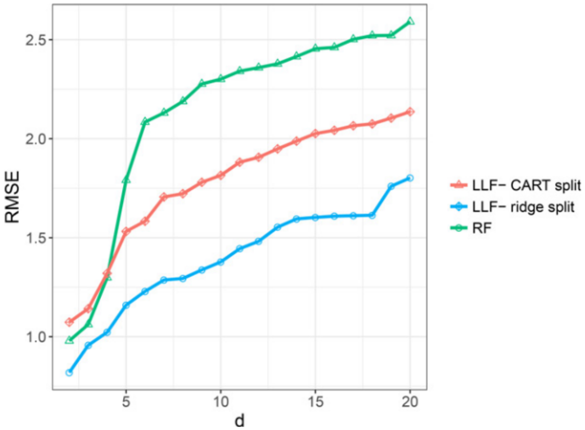


Figure 4. Results from testing different splitting rules on data generated from Equation (8). Here the *x*-axis is dimension *d*, varying from 2 to 20, and we plot the root mean square error of prediction from random forests and from local linear forests with CART splits and with the ridge residual splits. We let $n = 600$ and check results on 600 test points at 50 runs for each value of *d*.

The GRF package estimates locally linear random regression forests using the `ll_regression_forest` command.

The GRF package estimates locally linear random regression forests using the `ll_regression_forest` command.

It also estimates a linear causal forest using the `lm_forest` command. The method is explained in Section 3 of FTAW.

The GRF package estimates locally linear random regression forests using the `ll_regression_forest` command.

It also estimates a linear causal forest using the `lm_forest` command. The method is explained in Section 3 of FTAW.

In the final stage, after the x_i have been partialed out, there are *two* ridge penalties, and thus two tuning parameters to be chosen.

The GRF package estimates locally linear random regression forests using the `ll_regression_forest` command.

It also estimates a linear causal forest using the `lm_forest` command. The method is explained in Section 3 of FTAW.

In the final stage, after the x_i have been partialed out, there are *two* ridge penalties, and thus two tuning parameters to be chosen.

FTAW notes that honesty can improve or worsen predictions.

The GRF package estimates locally linear random regression forests using the `ll_regression_forest` command.

It also estimates a linear causal forest using the `lm_forest` command. The method is explained in Section 3 of FTAW.

In the final stage, after the x_i have been partialed out, there are *two* ridge penalties, and thus two tuning parameters to be chosen.

FTAW notes that honesty can improve or worsen predictions.

- With small sample sizes and strong signals, honesty may hurt predictive performance.

The GRF package estimates locally linear random regression forests using the `ll_regression_forest` command.

It also estimates a linear causal forest using the `lm_forest` command. The method is explained in Section 3 of FTAW.

In the final stage, after the x_i have been partialed out, there are *two* ridge penalties, and thus two tuning parameters to be chosen.

FTAW notes that honesty can improve or worsen predictions.

- With small sample sizes and strong signals, honesty may hurt predictive performance.
- With large sample sizes and weak signals, honesty may improve performance.

The GRF package estimates locally linear random regression forests using the `ll_regression_forest` command.

It also estimates a linear causal forest using the `lm_forest` command. The method is explained in Section 3 of FTAW.

In the final stage, after the x_i have been partialed out, there are *two* ridge penalties, and thus two tuning parameters to be chosen.

FTAW notes that honesty can improve or worsen predictions.

- With small sample sizes and strong signals, honesty may hurt predictive performance.
- With large sample sizes and weak signals, honesty may improve performance.
- In the former case, local linear corrections can help mitigate the loss of expressive power due to honesty.

The GRF package estimates locally linear random regression forests using the `ll_regression_forest` command.

It also estimates a linear causal forest using the `lm_forest` command. The method is explained in Section 3 of FTAW.

In the final stage, after the x_i have been partialed out, there are *two* ridge penalties, and thus two tuning parameters to be chosen.

FTAW notes that honesty can improve or worsen predictions.

- With small sample sizes and strong signals, honesty may hurt predictive performance.
- With large sample sizes and weak signals, honesty may improve performance.
- In the former case, local linear corrections can help mitigate the loss of expressive power due to honesty.

The paper includes both asymptotic theory and simulation results.

One interesting simulation expands on (6), with $10 \leq d \leq 50$. Recall that the first predictor enters very nonlinearly, but all the rest enter linearly.

One interesting simulation expands on (6), with $10 \leq d \leq 50$. Recall that the first predictor enters very nonlinearly, but all the rest enter linearly.

There are two cases, one with $\sigma = 5$ and one with $\sigma = 20$. The second case is much noisier than the first. Variance matters!

One interesting simulation expands on (6), with $10 \leq d \leq 50$. Recall that the first predictor enters very nonlinearly, but all the rest enter linearly.

There are two cases, one with $\sigma = 5$ and one with $\sigma = 20$. The second case is much noisier than the first. Variance matters!

- Locally linear forests always perform well, especially for d large.

One interesting simulation expands on (6), with $10 \leq d \leq 50$. Recall that the first predictor enters very nonlinearly, but all the rest enter linearly.

There are two cases, one with $\sigma = 5$ and one with $\sigma = 20$. The second case is much noisier than the first. Variance matters!

- Locally linear forests always perform well, especially for d large.
- BART (without tuning) always performs poorly.

One interesting simulation expands on (6), with $10 \leq d \leq 50$. Recall that the first predictor enters very nonlinearly, but all the rest enter linearly.

There are two cases, one with $\sigma = 5$ and one with $\sigma = 20$. The second case is much noisier than the first. Variance matters!

- Locally linear forests always perform well, especially for d large.
- BART (without tuning) always performs poorly.
- Boosting (XGBoost) performs well when the variance is low but badly when it is high.

One interesting simulation expands on (6), with $10 \leq d \leq 50$. Recall that the first predictor enters very nonlinearly, but all the rest enter linearly.

There are two cases, one with $\sigma = 5$ and one with $\sigma = 20$. The second case is much noisier than the first. Variance matters!

- Locally linear forests always perform well, especially for d large.
- BART (without tuning) always performs poorly.
- Boosting (XGBoost) performs well when the variance is low but badly when it is high.
- Standard random forests (estimated using GRF, of course) perform badly when the variance is low but well when it is high.

One interesting simulation expands on (6), with $10 \leq d \leq 50$. Recall that the first predictor enters very nonlinearly, but all the rest enter linearly.

There are two cases, one with $\sigma = 5$ and one with $\sigma = 20$. The second case is much noisier than the first. Variance matters!

- Locally linear forests always perform well, especially for d large.
- BART (without tuning) always performs poorly.
- Boosting (XGBoost) performs well when the variance is low but badly when it is high.
- Standard random forests (estimated using GRF, of course) perform badly when the variance is low but well when it is high.

Thus variance affects boosting and random forests in opposite ways.

One interesting simulation expands on (6), with $10 \leq d \leq 50$. Recall that the first predictor enters very nonlinearly, but all the rest enter linearly.

There are two cases, one with $\sigma = 5$ and one with $\sigma = 20$. The second case is much noisier than the first. Variance matters!

- Locally linear forests always perform well, especially for d large.
- BART (without tuning) always performs poorly.
- Boosting (XGBoost) performs well when the variance is low but badly when it is high.
- Standard random forests (estimated using GRF, of course) perform badly when the variance is low but well when it is high.

Thus variance affects boosting and random forests in opposite ways.

They also combine lasso and random forests by running lasso on half the data, then using RF to model the residuals on the other half.

One interesting simulation expands on (6), with $10 \leq d \leq 50$. Recall that the first predictor enters very nonlinearly, but all the rest enter linearly.

There are two cases, one with $\sigma = 5$ and one with $\sigma = 20$. The second case is much noisier than the first. Variance matters!

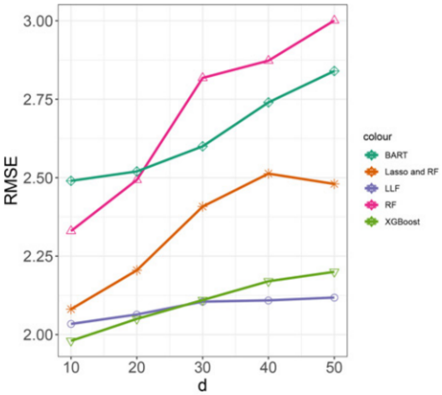
- Locally linear forests always perform well, especially for d large.
- BART (without tuning) always performs poorly.
- Boosting (XGBoost) performs well when the variance is low but badly when it is high.
- Standard random forests (estimated using GRF, of course) perform badly when the variance is low but well when it is high.

Thus variance affects boosting and random forests in opposite ways.

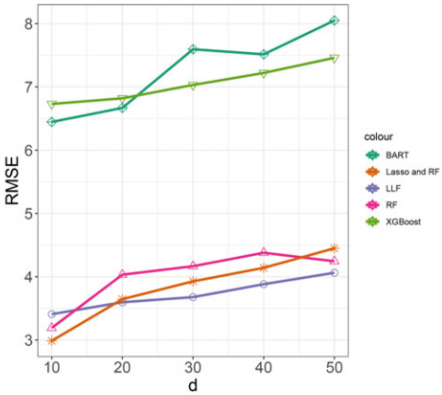
They also combine lasso and random forests by running lasso on half the data, then using RF to model the residuals on the other half.

This works well when the variance is high.

Figure 22.3 — Performance of several methods on simulated data



Errors on low variance



Errors on high variance