Journal of Machine Learning Research 26 (2025) 1-49 Submitted 2/24; Revised 4/25; Published 7/25

Randomization Can Reduce Both Bias and Variance: A Case Study in Random Forests

Brian Liu BRILIU@MIT.EDU
Operations Research Center
Massachusetts Institute of Technology
Cambridge, 02139-4307, USA

Rahul Mazumder RAHULMAZ@MIT.EDU
Operations Research Center and Sloan School of Management
Massachusetts Institute of Technology
Cambridge, 02139-4307, USA

Editor: Florence d'Alche-Buc

Abstract

We study the often overlooked phenomenon, first noted in Breiman (2001), that random forests appear to reduce bias compared to bagging. Motivated by an interesting paper by Mentch and Zhou (2020), where the authors explain the success of random forests in low signal-to-noise ratio (SNR) settings through regularization, we explore how random forests can capture patterns in the data that bagging ensembles fail to capture. We empirically demonstrate that in the presence of such patterns, random forests reduce bias along with variance and can increasingly outperform bagging ensembles when SNR is high. Our observations offer insights into the real-world success of random forests across a range of SNRs and enhance our understanding of the difference between random forests and bagging ensembles. Our investigations also yield practical insights into the importance of tuning *mtry* in random forests.

Keywords: Random Forests, Bagging, Ensemble Learning, Bias-Variance Tradeoff

1. Introduction

For over two decades, random forests (Breiman, 2001) have enjoyed widespread acclaim in machine learning and remain among the most competitive off-the-shelf supervised learning algorithms today. Random forest combine excellent predictive performance, comparable to boosting algorithms (Caruana and Niculescu-Mizil, 2006), with ease-of-use, since the algorithm is embarrassingly parallelizable and robust to overfitting. As such, random forests are applied in practice across a variety of disciplines that range from finance (Liu et al., 2015) to bioinformatics (Díaz-Uriarte and Alvarez de Andrés, 2006).

Remarkably, random forests are a simple modification of the original bootstrap aggregating (bagging) algorithm proposed by Breiman (1996) to construct tree ensembles. In bagging, a dataset is sampled with replacement repeatedly and a decision tree is fit on each sample. The sampled datasets, which we refer to as bags, are typically of the same size as the original dataset. The predictions of the trees are combined across bags by averaging (regression) or voting (classification) to produce the final prediction of the ensemble. Random forests

repeat this procedure with just a single modification. For each split in each tree in the ensemble, only a random subset of $mtry \in (0, 1)$ fraction of the variables in the dataset are considered when constructing the split. This simple modification can increase the predictive performance of the random forest compared to bagging, as empirical experiments in Breiman (2001) and Caruana and Niculescu-Mizil (2006) show.

Many empirical explanations for the success of bagging tree ensembles have been proposed in literature. Classically, Friedman (1997) argues that bagging reduces the variance of an ensemble without substantially increasing ensemble bias. Domingos (1997), on the other hand, proposes that bagging approximates the optimal procedure for Bayesian model averaging by sampling from the model space. Bühlmann and Yu (2002) argues that bagging softens the hard decision surfaces generated by trees, which reduces both ensemble bias and variance. Grandvalet (2004) proposes an orthogonal explanation, that bagging equalizes the influence of high leverage points in the data. Sampling with replacement ensures that a single high-leverage point appears in just two-thirds of the generated bags. More recently, Wyner et al. (2017) argues that bagging ensembles and by extension random forests, work well because they interpolate the training data while isolating the effects of noisy data points.

In stark contrast, few empirical explanations exist for why *additional* randomization in a random forest improves ensemble performance over bagging, and the main focus of our paper is to investigate this phenomenon. Canonically, Breiman (2001) suggests that randomizing the features considered per split reduces the correlation between trees, which in turn further reduces the variance of the ensemble. A textbook example from Hastie et al. (2009) illustrates this effect well; the variance of the average of B i.i.d random variables (trees) each with variance σ^2 is given by:

$$\gamma\sigma^2 + \frac{1-\gamma}{B}\sigma^2, \tag{1}$$

where γ is the pairwise correlation between variables. Given a sufficiently large number of trees B, the variance of the average is dominated by the first term. So reducing pairwise correlation reduces variance. While variance reduction in random forests is well understood, Breiman suggests that other factors may contribute to the success of random forests over bagging. In the concluding remarks of Breiman (2001), Breiman mentions that the accuracy of random forests indicates that they "act to reduce bias" but that the "mechanism for this is not obvious." However, this notion that random forests can reduce bias compared to bagging appears to have been largely overlooked in subsequent works. In fact, the second edition of the canonical textbook *The Elements of Statistical Learning* (Hastie et al., 2009) states in Chapter 15 that "the only hope of improvement [of random forests and bagging] is through variance reduction." As such, the idea that random forests improve performance solely through variance reduction appears to be widely accepted in statistical lore.

Building on the idea that random forests primarily benefit from variance reduction, a recent interesting paper by Mentch and Zhou (2020) proposes the novel argument that randomization in random forests reduces the effective degrees of freedom (DoF) of the model relative to bagging, and, as such, regularizes the model to outperform bagging ensembles in low signal-to-noise (SNR) ratio settings. The authors demonstrate that this regularization effect occurs through simulation studies and semi-synthetic experiments.

Motivated by the interesting arguments in Mentch and Zhou (2020), we seek to better explain the success of random forests when SNR is high. In our view, real-world problems

span a wide range of SNRs (Mazumder, 2020) and random forests work well in applications that range from financial modeling, where SNR is typically low, (Booth et al., 2014; Lohrmann and Luukka, 2019; Baba and Sevil, 2020) to signal processing where SNR can be quite high (Cerrada et al., 2016; Xu et al., 2023; Saki et al., 2016). A complementary explanation for the success of random forests may be required to account for the popularity of the algorithm in high SNR fields.

In this paper, we revisit the widely-held belief that random forests *only* reduce variance and show how random forests can reduce bias compared to bagging, which helps explain their success in high SNR settings. We introduce and formalize the notion of *hidden patterns*, patterns in the data that bagging ensembles have difficulty capturing. By randomly selecting subsets of features to consider per split, random forests better capture these patterns, which in turn reduces ensemble bias. To provide intuition on hidden patterns, we show an example below.

Hidden Pattern Motivating Example

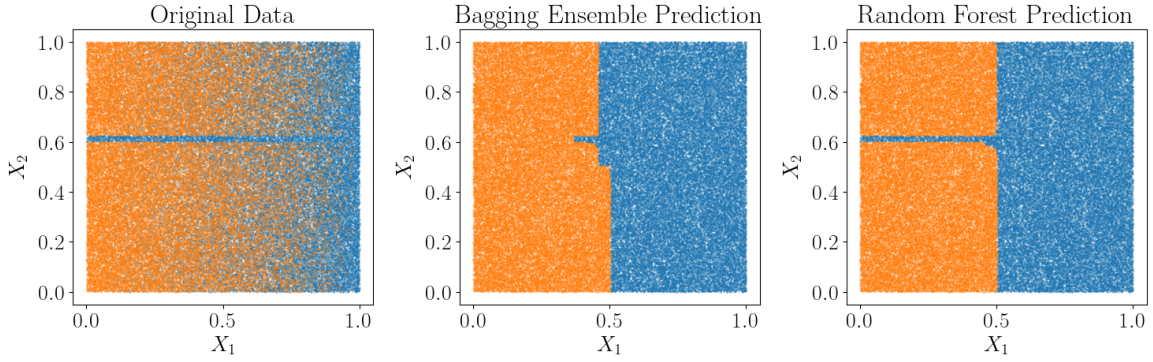


Figure 1: The relationship between X_2 and Y is hidden from a bagging ensemble by X_1 due to the myopic nature of how splits in a decision tree are constructed.

We consider a 2-dimensional grid of points (X_1, X_2) where $X_1, X_2 \in [0, 1]$ and uniformly generate 10000 points within this grid. We assign these points to classes $Y \in \{0, 1\}$ following these rules. All points with $0.6 \leq X_2 \leq 0.65$ are assigned to the positive class ($Y = 1$) with probability 0.9. All other points are assigned to the positive class with probability X_1.

The leftmost plot in Figure 1 shows a visualization of this data, with positive class points marked in blue and negative class points marked in orange. We see from this plot that we need two splits to capture the relationship between X_2 and Y: one on $X_2 \leq 0.65$ and one on $X_2 \geq 0.6$. The relationship between X_1 and Y, on the other hand, can be fairly well captured by a single split down the middle: $X_1 \leq 0.5$.

The relationship between X_2 and Y is an example of a hidden pattern. Due to the myopic nature of how splits are constructed, decision trees in a bagging ensemble will typically split on X_1 before X_2, and, as a result, the relationship between X_2 and Y may be hidden from the bagging ensemble by X_1. The middle plot in Figure 1 shows the predictions/decision boundaries of a bagging ensemble trained on the data, and we observe that this ensemble fails to capture the relationship between X_2 and Y. Random forests, however, can uncover

hidden patterns by randomly selecting subsets of features to consider per split. As we can see from the rightmost plot in Figure 1, the decision boundaries of a random forest fit on the data better capture the relationship between X_2 and Y compared to bagging ensembles. Later in this paper, we formally define hidden patterns and empirically demonstrate how random forests can reduce the training error and bias of the model through uncovering these patterns.

Main Contributions

We highlight our main contributions below.

- We revisit the popular belief that random forests *solely* reduce variance and we show how random forests can reduce bias.
- We introduce and describe the notion of hidden patterns, relationships in the data that bagging ensembles have difficulty capturing, and show that random forests better capture these relationships by randomly selecting the subset of features considered per split.
- We show that on datasets with hidden patterns, random forests increasingly outperform bagging ensembles as SNR increases. This finding complements the arguments presented in Mentch and Zhou (2020) by offering additional insight into the success of random forests over bagging when SNR is high.
- We empirically demonstrate that random forests reduce bias in addition to variance relative to bagging on datasets with hidden patterns.
- Our investigation yields the important practical takeaway that *mtry* should be tuned. We explore how *mtry* controls the ability of a random forest to reduce bias and analyze how this parameter is sensitive to non-informative features in the data.

2. Preliminaries

We first introduce notation and provide an overview of bagging ensembles and random forests. We then explore in detail the main arguments in Mentch and Zhou (2020): that random forests reduce effective DoF compared to bagging, which allows them to increasingly outperform bagging ensembles as SNR decreases. Interestingly, we show here that trimming trees in a bagging ensemble can reduce effective DoF by an equivalent degree to that of random forest, *without* improving performance. This finding suggests that random forests have additional advantages over bagging beyond reducing effective DoF, motivating our investigation into how random forests reduce bias.

2.1 Notation and Background

Let $X \in \mathbb{R}^{n \times p}$ represent our data matrix of n rows and p columns and let $Y \in \mathbb{R}^n$ represent the prediction target. Let $X^{(i)}$ denote the i^{th} row (observation) in data matrix X and let X_j denote the j^{th} column (feature). Assume that we have some, potentially unknown, data-generating procedure that can be represented by function $f : \mathbb{R}^{n \times p} \to \mathbb{R}^n$ that operates

on data matrix X. The relationship between X and Y is given by $Y = f(X) + \epsilon$ where $\epsilon \in \mathbb{R}^n$ is a vector of errors.

Bagging ensembles and random forests are made up of decision trees, and a tree $T(X)$ maps data matrix X to prediction vector $\hat{f}(X) \in \mathbb{R}^n$. Trees consist of internal nodes and leaf nodes. Each internal node represents a split of the feature space on a single feature, and the leaf nodes collectively partition the entire space. Trees are grown greedily, from root to leaf, and each split is selected to minimize the impurity of the directly resulting nodes (Breiman, 2017). Each leaf node assigns a prediction value to the corresponding partition of the feature space, typically the mean of the training data points in the partition for regression or the majority class for classification. Decision trees are highly flexible models; growing a tree to full-depth, where each leaf node contains just a single data point, can produce a model with 0 training loss. However, full-depth trees exhibit high variance and may perform poorly out-of-sample (Breiman, 2017; Hastie et al., 2009).

Bagging ensembles follow this procedure to reduce variance. We generate $X^1, \ldots, X^m \in \mathbb{R}^{n \times p}$ datasets (bags) by sampling with replacement rows of X. We fit a decision tree $T^k(X)$ on each bag and the trees are averaged to output the final prediction (for regression):

$$\hat{f}(X) = \frac{1}{m} \sum_{k=1}^{m} T^k(X).$$

To form a random forest, we add the restriction that when building each tree, we only consider $mtry < 1.0$ proportion of the features when constructing each split. We refer to this procedure as **split feature subsetting (SFS)**.

We make the distinction between bagging ensembles and random forests that random forests are fit with $mtry < 1.0$ and bagging ensembles are fit with $mtry = 1.0$, and we emphasize that this leads to estimators with very different operating characteristics. To highlight this distinction, we refer to random forests as SFS ensembles throughout this paper. In other works, random forests are often defined to have $mtry \leq 1.0$, and bagging ensembles are viewed as a special case of random forests with $mtry = 1.0$. Since our paper focuses on the comparison between random forests and bagging we take this additional step to distinguish the two procedures.

As an aside, we briefly discuss tree size in random forests. In the original random forest paper, Breiman recommends growing decision trees to their full depth, where each leaf node contains just a single observation (Breiman, 2001; Zhou and Mentch, 2023). This recommendation is followed by popular software implementations of the random forest algorithm (Liaw and Wiener, 2002; Pedregosa et al., 2011). However, subsequent works suggest that tuning tree depth has the potential to improve the predictive performance of a random forest when SNR is low (Zhou and Mentch, 2023; Lin and Jeon, 2006). Since we are interested in bias reduction in random forests when SNR is high, we focus primarily on random forests with full-depth trees.

2.2 Random Forest and Effective Degrees of Freedom

Mentch and Zhou (2020) in their interesting paper empirically demonstrate that reducing $mtry$, which controls the proportion of features to consider in SFS, reduces the effective degrees of freedom (DoF) of the ensemble (Efron and Tibshirani, 1986). Here, we note that

procedures other than SFS can reduce the effective DoF of bagging ensembles. For example, Mentch and Zhou (2020) (Section 3) mentions that limiting the maximum number of leaf nodes per tree (*maxnodes*), a procedure we will refer to as **TRIM**, naturally reduces effective DoF. In the next section, we empirically demonstrate that while SFS ensembles outperform bagging when SNR is low, TRIM ensembles, tuned to have the same effective DoF as SFS ensembles, do not outperform bagging under any of the SNR regimes in the experiments in Mentch and Zhou (2020). Consequently, we believe that SFS randomization in random forests goes beyond reducing effective DoF and model variance, and may in fact reduce model bias compared to bagging. Exploring the mechanisms behind this bias reduction in random forests is the main focus of our paper.

For consistency, we use the same procedure discussed in Mentch and Zhou (2020) and Efron and Tibshirani (1986) to estimate effective DoF. Let function $\hat{f}$ produce estimates $\hat{y}_1, \ldots, \hat{y}_n$, and assume that these estimates differ from the true values $y_1, \ldots, y_n$ by errors $\epsilon_1, \ldots, \epsilon_n$, which are independent Gaussian $N(0, \sigma^2)$. The effective DoF of function $\hat{f}$ is given by:

$$\mathrm{df}(\hat{f}) = \frac{1}{\sigma^2} \sum_{i=1}^{n} \mathrm{Cov}(y_i, \hat{y}_i).$$

Following the approach in Mentch and Zhou (2020), we estimate this using Monte Carlo simulation.

2.2.1 Case Study: TRIM vs. SFS

We extend the simulation used in Mentch and Zhou (2020) (Section 4) and generate synthetic data using the following setup. We generate data matrix $X \in \mathbb{R}^{n \times 5}$ where $n \in \{200, 1000\}$ and each entry of X is sampled from a uniform $U(0, 1)$ distribution. We generate response Y via the following models:

$$Y = 10 \sin(\pi X_1 X_2) + 20(X_3 - 0.05)^2 + 10X_4 + 5X_5 + \epsilon, \tag{2}$$

and

$$Y = 0.1e^{4X_1} + \frac{4}{1 + e^{-20(X_2 - 0.5)}} + 3X_3 + 2X_4 + X_5 + \epsilon. \tag{3}$$

As in Mentch and Zhou (2020), we refer to (2) as the MARS model and (3) as the MARSadd model in reference to their first appearance in Friedman (1991). The error term ϵ follows an independent Gaussian $N(0, \sigma^2)$ distribution and σ^2 is prespecified to determine the signal-to-noise ratio of the generated data. Recall that SNR is defined by:

$$\mathrm{SNR} = \frac{\mathrm{Var}\big(f(X)\big)}{\mathrm{Var}(\epsilon)}. \tag{4}$$

We first generate synthetic datasets of size $n \in \{200, 1000\}$ using the MARS and MARSadd functions and vary the SNR of the dataset on 10 evenly log-spaced points from 0.042 to 6. To replicate the results from Mentch and Zhou (2020), we split the data into a 50%-50% train-test split and fit 2 models: a random forest with $mtry = 0.33$ and a bagging

ensemble with $mtry = 1.0$. For both models, we grow 100 trees and set $maxnodes = 200$[1]. We repeat this procedure for 500 trials across each SNR level to obtain estimates of the test error and effective DoF for each model. Finally, we repeat this entire procedure while fitting another bagging ensemble of 100 trees with TRIM and *maxnodes* set such that the effective DoF of the TRIM ensemble matches the effective DoF of the SFS random forest. We discuss the results of this simulation below.

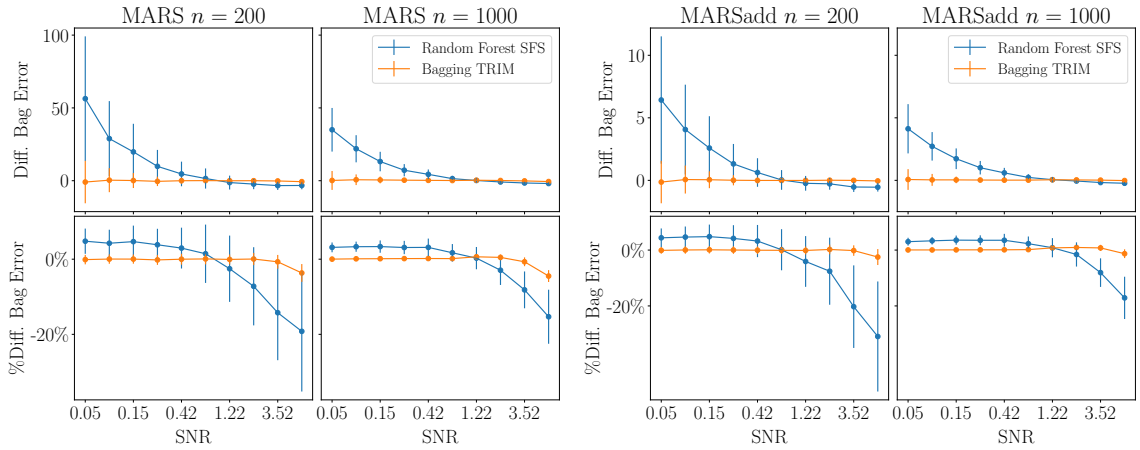


Figure 2: Even after tuning TRIM to match the effective DoF of SFS, TRIM does not improve ensemble performance at any SNR level.

The plots in Figure 2 summarize the results of our simulation study. In the top row of plots, we report the performance of random forest SFS and bagging TRIM relative to vanilla bagging. The vertical axis shows the difference between the errors of the bagging ensemble and the errors of the SFS/TRIM ensemble and the horizontal axes show SNR. The SFS error curves (blue lines) are similar to the ones found in Mentch and Zhou (2020) (Section 4, Figure 4) and show that the relative performance of SFS random forests increases as SNR decreases. The TRIM error curves (orange lines) show that TRIM does not improve out-of-sample performance relative to bagging. We also show the percent difference in error between bagging and SFS/TRIM in the bottom row of plots, since we expect raw errors to be higher when SNR is low. The vertical axis shows:

$$\frac{\text{Error(Bagging)} - \text{Error(SFS/TRIM)}}{\text{Error(bagging)}} \times 100\%,$$

and the horizontal axes again show SNR. Our conclusions remain the same, the relative performance of SFS over bagging increases inversely with SNR but the relative performance of TRIM compared to bagging remains the same across all SNR values.

Recall that the TRIM bagging ensembles were tuned to match the effective degrees of freedom (DoF) of the SFS random forests. While both TRIM and SFS reduce effective DoF, only SFS improves ensemble performance in this example. This suggests that factors beyond DoF reduction and variance reduction contribute to the effectiveness of SFS in random forests.

1. We set the maximum number of leaf nodes to 200 so that we are consistent with Mentch and Zhou (2020).

We find this result particularly interesting, as it aligns with an observation first made by Breiman (2001). Breiman noted that alternative forms of randomization—such as randomizing split selection[2] or adding random noise to the outputs of individual decision trees—can improve predictive performance to some extent. However, none of these methods match the performance gains achieved through SFS randomization over the features considered at each split (see Section 3 of Breiman, 2001). This indicates that there is something distinctive about SFS randomization, which motivates our investigation into its role in bias reduction.

2.2.2 TRIM vs. SFS Additional Discussions

As an aside, in §A.1 of the appendix we report the tuned *maxnodes* values for our TRIM ensemble. It is interesting to note that for medium to high SNRs, the tuned TRIM ensemble has much fewer leaf nodes compared to the bagging ensemble, even though the predictive performances of the models are similar. This is consistent with the understanding that tuning tree depth seldom affects predictive performance when SNR is high (Hastie et al., 2009).

We note that in other cases, trimming tree depth in random forests has been shown to improve predictive performance through regularization when SNR is low (Zhou and Mentch, 2023). Various methods have been proposed to reduce the effective degrees of freedom of random forests and bagging ensembles (Mentch and Zhou, 2022, 2020; Zhou and Mentch, 2023), and these regularization techniques yield different impacts on predictive performance. For example, Mentch and Zhou (2020) proposes augmented bagging, a technique to regularize random forests by adding noise features to the training data. We show in §A.2 of the appendix that similar to TRIM, augmented bagging can reduce the effective DoF of the model without improving performance in the examples discussed above.

Also, as a quick detour, we briefly switch to the classification setting to visually highlight the difference between TRIM and SFS. On a 3-dimensional space (X_1, X_2, X_3), consider a sphere of radius 1 with center at $(0.5, 0.5, 0.5)$. We uniformly generate $n = 10000$ data points and assign points inside of the sphere to positive class $Y = 1$ with probability $p = 0.9$. We assign points outside of the sphere to positive class $y = 1$ with probability $p = 0.1$; positive class points outside of the sphere and negative class $Y = 0$ points inside of the sphere can be viewed as noise points. Our goal is to test how well bagging ensembles, TRIM bagging ensembles, and SFS random forests recover the decision boundary of the sphere.

The leftmost plot in Figure 3 shows the actual data. The blue points represent $y = 1$ and the orange points represent $y = 0$. The 3 right plots in the figure show the predictions of the bagging, TRIM, and SFS random forest ensembles, with $\hat{y} = 1$ in blue and $\hat{y} = 0$ in orange. We observe that the bagging ensemble and SFS random forest recover the true spherical decision boundary of the underlying data. The decision boundary of the TRIM ensemble, however, has degraded into a cube. While TRIM and SFS both reduce effective DoF, they impact the model very differently. Here, TRIM appears to significantly reduce the ability for the ensemble to fit the original training data, while SFS does not. In the following sections, we return to the regression setting to investigate how SFS randomization can in fact *improve* the ability for an ensemble to fit the training data, which leads to a reduction in model bias.

2. Randomly selecting splits instead of randomizing the features considered per split

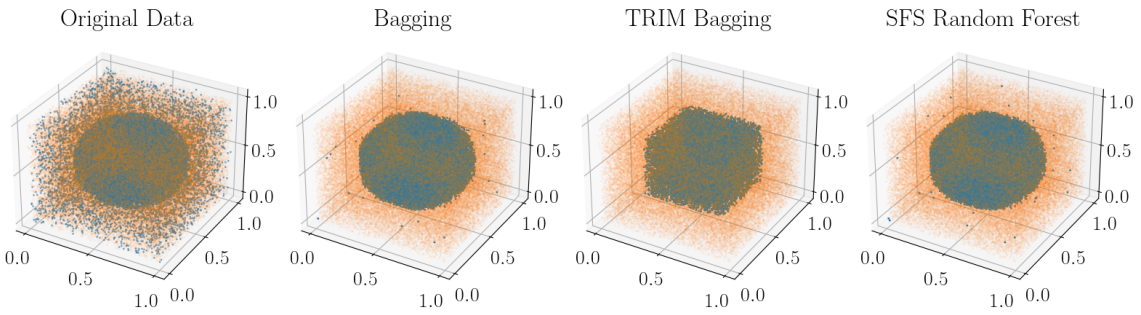


Figure 3: Both TRIM and SFS reduce the effective DoF of the model, however, their effects on the decision boundary of the ensemble differ drastically.

3. Random Forests and Hidden Patterns

In the next two sections, we present the main investigations of our paper and show that random forests can reduce both bias and variance on datasets with hidden patterns. We start by discussing an extended case study in two dimensions to visualize how random forests can better fit the training data compared to bagging ensembles, in certain situations. Using this example, we challenge the existing statistical lore that random forests *only* reduce variance compared to bagging. We then characterize our notion of hidden patterns and empirically show that on datasets with such patterns random forests reduce both bias and variance.

3.1 Case Study: A Hidden Pattern in 2D

We present an easy-to-visualize example in two dimensions of how random forests can uncover patterns in the data missed by bagging. Consider the model below, which we refer to as **Hidden2D**, with the data generating procedure (DGP):

$$Y = X_1 - \mathbb{1}(0.6 \leq X_2 \leq 0.65) + \epsilon. \tag{5}$$

We say that Hidden2D is generated from function $f(X) = f_1(X_1) + f_2(X_2)$, where $f_1(X_1) = X_1$ and $f_2(X_2) = -\mathbb{1}(0.6 \leq X_2 \leq 0.65)$. The DGP function $f(X)$ operates on the data matrix X while the DGP functions $f_1(X_1)$ and $f_2(X_2)$ operate on individual columns of X. We show $f_1(X_1)$ and $f_2(X_2)$ in the two leftmost displays in Figure 4. Following the procedure from the simulation studies presented in Mentch and Zhou (2020) (Sections 3 and 4), we generate the entries of data matrix $X \in \mathbb{R}^{n \times 2}$ from a uniform $U(0, 1)$ distribution, and in the rightmost scatter plot in Figure 4 we show an example of the Hidden2D model with $n = 1000$ observations and ϵ, the independent Gaussian noise vector, set such that the SNR of the model is equal to 6. Each point in the scatter plot represents a observation and the color of the point indicates its corresponding Y value.

Since the entries of X are generated from a $U(0, 1)$ distribution, the DGP function on feature X_2, $\mathbb{1}(0.6 \leq X_2 \leq 0.65)$, only affects around 5% of the observations in the data. This is shown in the rightmost plot of Figure 4 by the thin horizontal band in dark blue, located right above $X_2 = 0.6$. As we show in the section below, decision trees often have trouble capturing this relationship between feature X_2 and Y.

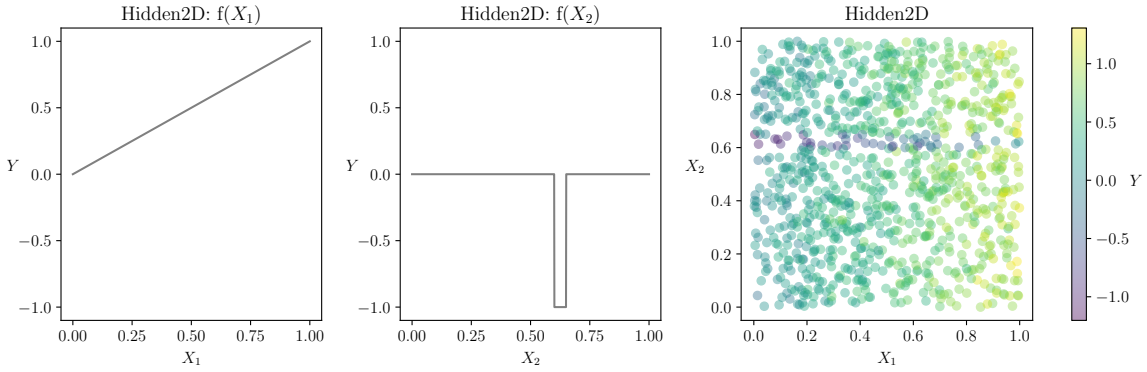


Figure 4: The left and middle display show $f_1(X_1)$ and $f_2(X_2)$, and the rightmost display shows an example of the Hidden2D model with 1000 observations and a SNR of 6.

3.1.1 Visualization: Hidden2D Single Decision Tree

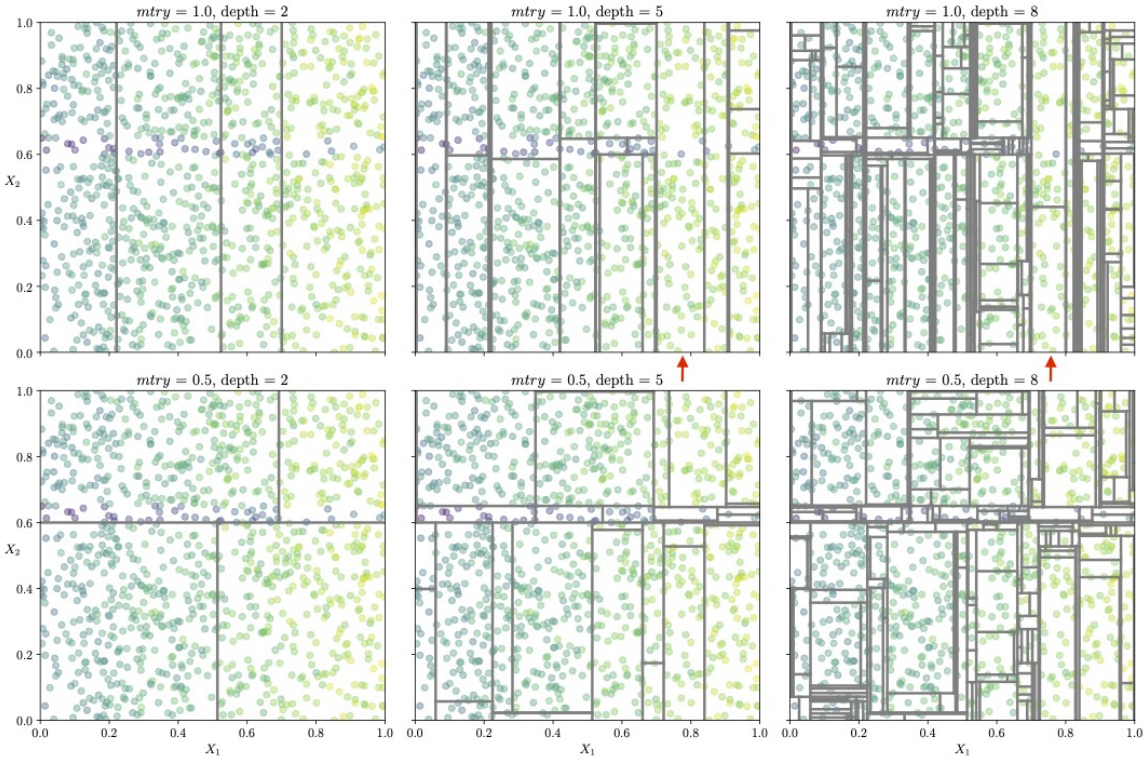


Figure 5: Splits of a decision tree fit on the Hidden2D example at various depths. The rectangular regions delineates the boundaries of the nodes of the tree at each depth level level. The top row of plots shows a tree with $mtry = 1.0$, the bottom row shows a tree with $mtry = 0.5$.

In the top row of plots in Figure 5, we fit a *single* decision tree on the Hidden2D example from above, with 1000 observations and a SNR of 6. The grey lines show the splits constructed by the tree, vertical lines are splits on feature X_1 and horizontal lines are splits on feature X_2. Each plot shows all of the splits in the tree up to a certain depth level, depth $\in \{2, 5, 8\}$. We observe that the decision tree tends to split on feature X_1 before X_2; the leftmost plot shows that the first 3 splits of the tree, of depths 1 and 2, are only on feature X_1. This makes sense since decision trees are constructed greedily, with the best split selected to minimize the error of the directly resulting partitions. The DGP function on feature X_1, $f_1(X_1) = X_1$, affects all of the observations while the DGP function on X_2 affects only a small subset ($\sim$5%) of observations, so initially splitting on feature X_1 reduces error by a greater extent than splitting on X_2.

Importantly, we note that the subsequent splits on feature X_2 are constructed with respect to these initial splits on X_1, due to the hierarchal nature of decision trees. Here, we observe that it is difficult for the horizontal splits on X_2 to capture the underlying DGP function, $f_2(X_2) = -\mathbb{1}(0.6 \leq X_2 \leq 0.65)$, when conditioned on the vertical splits on X_1. The red arrows in Figure 5 show the region where this is most apparent. Consider the middle display in the top row of Figure 5, this plot shows the splits of our decision tree at depth 5. The red arrow in this display shows the region delineated by the splits $0.70 \leq X_1 \leq 0.81$, and, within this region, it is hard to construct a horizontal split on X_2 that captures DGP function $f_2(X_2)$. As we can see from the rightmost display in the top row of Figure 5, our decision tree still fails to capture this pattern by depth level 8.

In this example, the relationship between feature X_2 and Y is hidden by X_1. We refer to this relationship as a hidden pattern, a notion we formally define in §4. For now, consider the bottom row of plots in Figure 5. These plots show a single decision tree fit on the same example from above, however, each split is constructed on a *randomly* selected feature, i.e., *mtry* is set to 0.5 for this decision tree. We can see from the leftmost display that this leads to earlier splits on feature X_2. As a result, this tree with $mtry = 0.5$ can better capture the DGP function, $f_2(X_2) = -\mathbb{1}(0.6 \leq X_2 \leq 0.65)$, compared to the $mtry = 1.0$ tree shown in the row above. Below, we present a simulation study to show how the effect discussed in this section extends to bagging ensembles and random forests.

3.1.2 Hidden2D Simulation Study: Bagging and SFS Random Forests

Using the Hidden2D model (5), we again generate $n = 1000$ data points while setting ϵ such that the SNR of the data is equal to 6. Following the setup presented in §2.2.1, we split the data via a 50-50 train/test split and construct two ensembles of 500 **full-depth** trees on the train split: a bagging ensemble and a SFS random forest with $mtry = 0.5$. In these full-depth tree ensembles, each tree is grown so that each leaf node contains just a single observation. We compare the test performances of the two ensembles. The bagging ensemble has a test MSE of 0.047 and, interestingly, the SFS random forest has a test MSE of 0.029, a 37% decrease in test error.

We find it unlikely that this observation can be fully explained by the variance reduction that SFS random forests achieve over bagging. Since random forests reduce effective DoF relative to bagging, we would expect them to *increase* test error when SNR is high, if the advantage of random forests over bagging were solely due to variance reduction. Rather, we

believe that this observed decrease test error in an high SNR setting can be explained by the biases of the models. In this example, the *training* MSE of the SFS random forest is reduced to 0.005 from 0.007 for bagging, a 28.6% decrease, which suggests a reduction in bias.

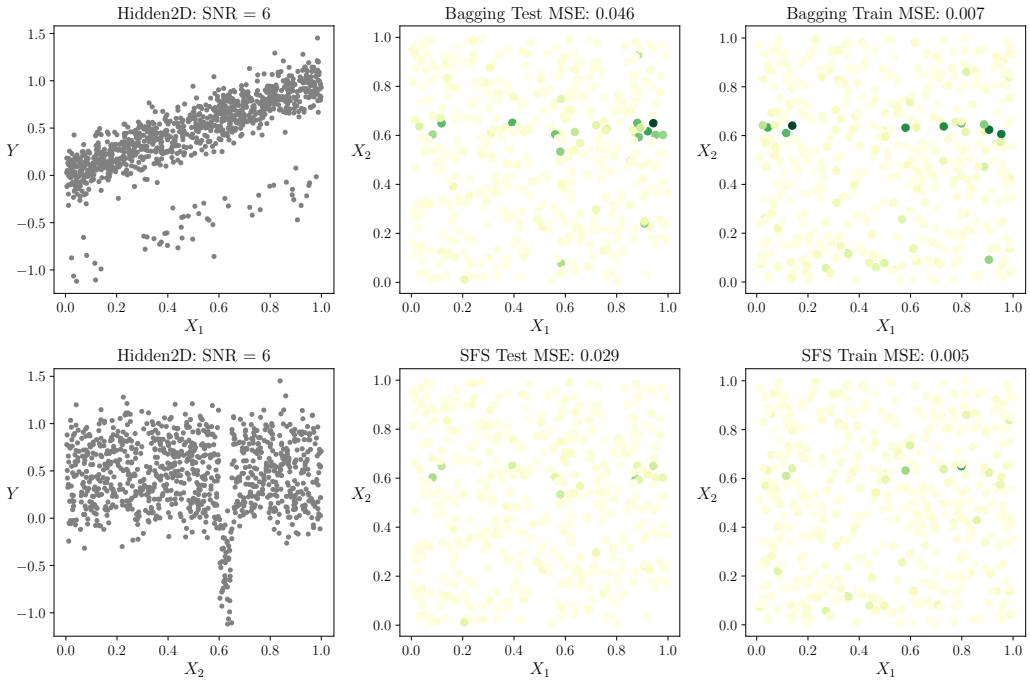


Figure 6: SFS random forests and bagging ensembles fit on Hidden2D generated data with high SNR. SFS allows random forests to better capture the effect of the hidden patterns.

To understand how random forests can reduce bias in this example, we visualize the distribution of errors for each point in the training set and test set. In the center column of Figure 6, we show scatter plots of the test data points with X_1 on the horizontal axes and X_2 on the vertical axes. The color gradient shows the squared test error for each data point, darker colors indicate higher errors. The top plot shows the error distribution of the bagging ensemble and the bottom plot shows the error distribution of the SFS random forest and the color gradient is normalized across all plots. The right column in Figure 6 shows these same plots but for the training data and training error.

We observe from these plots that for bagging ensembles (top two plots) data points in the $0.6 \leq X_2 \leq 0.65$ horizontal band have higher training and test errors. This band corresponds to the region of X_2 where the DGP function $f_2(X_2) = -\mathbb{1}(0.6 \leq X_2 \leq 0.65)$ is greater than zero. We also see that in this region, SFS (bottom two plots) reduces both training and test error relative to bagging. We see from this example that the bagging ensemble is unable to capture the hidden pattern in feature X_2. The randomization from SFS better allows random forests to capture this effect, which can reduce the bias of the ensemble.

Bagging ensembles have trouble capturing the hidden pattern in X_2 due to the greedy nature of how decision trees are fit. In the full-depth decision trees of the bagging ensemble, feature X_1 typically dominates the top layers of the tree, as we see in Figure 7. As we discuss in the sections above, in the Hidden2D example it is difficult for the subsequent splits on

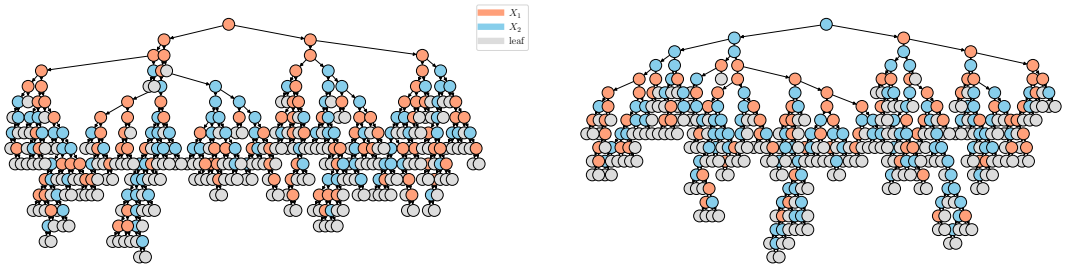


Figure 7: Visualization of a tree used in bagging ensemble (left) and SFS random forest (right). Feature X_1 dominates the top layers of the bagging ensemble tree. The color of each node shows the feature used in the split, orange for feature X_1 and blue for feature X_2. The grey nodes are leaves.

feature X_2 to capture the DGP function $f_2(X_2)$, when conditioned on the preceding splits on feature X_1. In SFS random forests, on the other hand, we force half of the splits in each tree to be on X_2. In doing so we capture the effect of the hidden pattern.

3.1.3 Hidden2D Simulation Study: Varying SNR

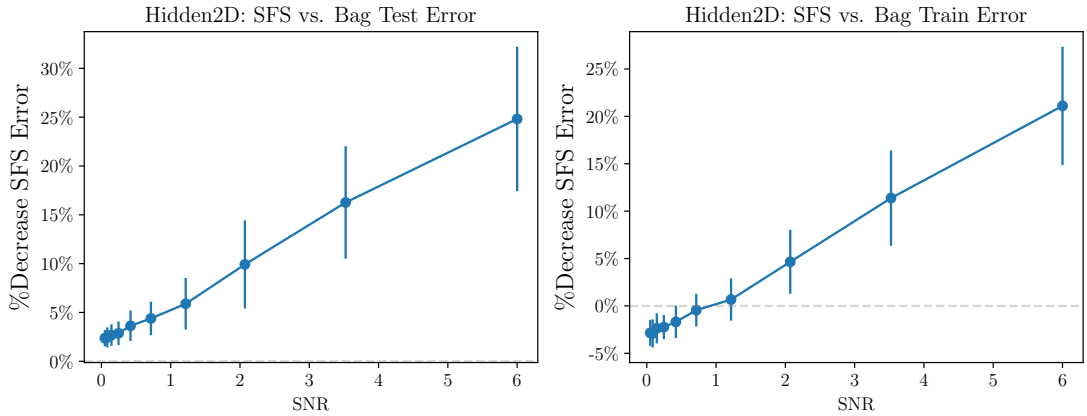


Figure 8: As SNR increases in this example, we observe that SFS increasingly outperforms vanilla bagging. SFS can capture the hidden pattern in feature X_2, which is more prominent in higher SNR settings.

For thoroughness, we evaluate how SFS random forests perform compared to bagging on the Hidden2D model across different SNR levels. We follow the experimental procedure from §2.2.1 and show the results in Figure 8. The left plot compares the test performance of SFS against bagging ensembles. The horizontal axis shows SNR and the vertical axis shows the percent decrease in test MSE between SFS and bagging. We observe that SFS outperforms bagging across all SNR ranges in this example; the percent decreases in test MSE are all positive. We see that as SNR increases SFS outperforms bagging by a larger degree.

The right plot in Figure 8 shows the percent decrease in *training* error between SFS random forests and bagging ensembles. We observe a positive trend between the percent decrease in training error and SNR, and we note that in high SNR settings, random forests have lower training errors than bagging ensembles. This suggests in this example that random forests reduce bias compared to bagging when SNR is high.

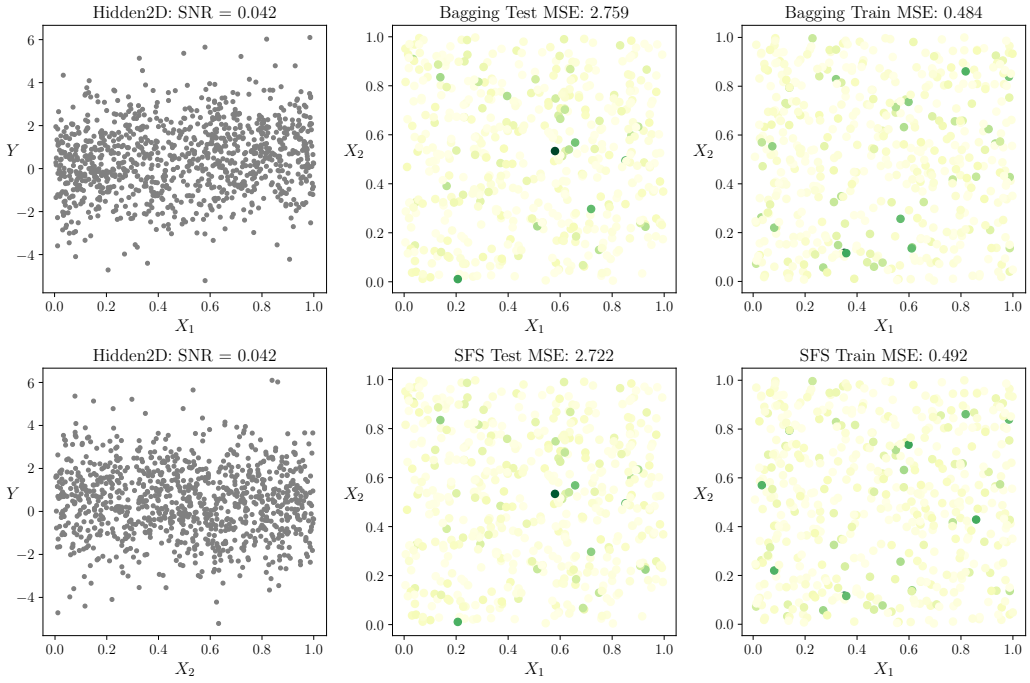


Figure 9: SFS random forests and bagging ensembles fit on Hidden2D generated data with low SNR. In low SNR settings, the effect from hidden pattern X_2 is obscured by noise.

We also notice that in the low SNR < 1 setting, SFS random forests increase training error relative to bagging; the percent decreases in training error here are negative. In this low SNR setting SFS still improves test performance over bagging, as shown in the left plot. This can be explained by the arguments in Mentch and Zhou (2020), that randomization in random forests acts as regularization in low SNR settings by reducing effective DoF. In this example, however, the improvement of SFS random forests over bagging is more pronounced when SNR is high. We see in this example that the hidden pattern in X_2 is obscured by noise when SNR is low. The left column of Figure 9 shows Y plotted against X_1 and X_2 when SNR $= 0.042$, and we do not observe the hidden pattern that was apparent in Figure 6, when SNR $= 6$. We also see that the distributions of the training and test errors are no longer clustered around the $0.6 \leq X_2 \leq 0.65$ horizontal band. In this case, the hidden pattern in X_2 is irrelevant considering the noise in the data, and as such, the improvement of SFS random forests over bagging is less pronounced.

Our findings build on the arguments in Mentch and Zhou (2020). While randomization acts as a form of regularization in low SNR settings, SFS randomization can enable random forests to better fit the training data on certain datasets compared to bagging. As a result,

random forests can outperform bagging when SNR is high. In the next section, we empirically demonstrate that this improvement in the high SNR = 6 setting stems from bias reduction.

3.1.4 Hidden2D Simulation Study: Bias Variance Decomposition

In our simulation study above, the underlying data generating procedure is known, and, as such, we can compute the decomposition of the error of our model into a bias term, variance term, and noise term. We follow the general bias-variance decomposition procedure discussed in Domingos (2000). Given regression data-generating procedure $Y = f(X) + \epsilon$, where ϵ is the irreducible error, and prediction model $\hat{f}(X)$ is fit on X to predict Y, the squared error of our model can be decomposed into:

$$E\big[\big(Y - \hat{f}(X)\big)^2\big] = \big(E\big[f(X) - \hat{f}(X)\big]\big)^2 + E\big[\big(\hat{f}(X) - E\big[\hat{f}(X)\big]\big)^2\big] + \mathrm{Var}(\epsilon) \tag{6}$$

where the first term is squared bias, the second term is variance, and the third term is irreducible error. The expectations are taken with respect to randomness in the training data and in the prediction model, e.g., the bootstrap and *mtry* randomization in the random forest algorithm. We repeat our Hidden2D simulation study (with SNR = 6) 500 times to estimate the bias and variance of both the bagging ensemble and SFS random forest.

Hidden2D	Bias^2	Variance
SFS Random Forest	0.00235	0.00493
Bagging Ensemble	0.00525	0.00838

Table 1: Bias variance decomposition for Hidden2D simulation study with SNR = 6.

We show the results of this bias variance decomposition in Table 1 and we see that in the high SNR setting of our Hidden2D study, random forests reduce both bias and variance compared to bagging. It is important to note that this finding shows that random forests reduce bias in addition to variance, an observations that appears to have been overlooked in prior literature.

3.2 Do Random Forests *Only* Reduce Variance?

In this section, we revisit the classical belief that random forests only reduce variance compared to bagging. Indeed, the second edition of *Elements of Statistical Learning* states in Chapter 15, Section 4.2 that: "improvements in prediction obtained by bagging or random forests are solely a result of variance reduction." The authors argue that this is due to the fact that "the bias of bagged trees is the same as that of the individual (bootstrap) trees" and that randomization is unlikely to reduce the bias of an individual tree grown to full depth Hastie et al. (2009).

While randomizing splits can increase tree bias in certain cases, our Hidden2D example suggests that it can also reduce bias in others. In our example, SFS randomization *lowers* the bias of full-depth decision trees. We repeat the bias variance decomposition discussed in §3.1.4 and estimate the bias of each individual tree in both the SFS random forest and the bagging ensemble. The distribution of these biases across trees are shown in Figure 10. From this plot, it is apparent that the bias of trees in the random forest is significantly lower than

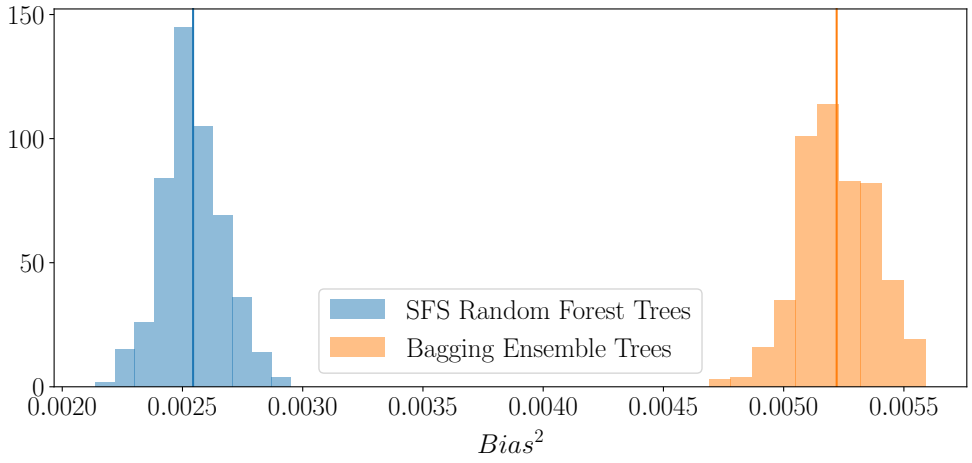


Figure 10: Distribution of the bias of individual trees in an SFS random forest and a bagging ensemble. Each tree is constructed off of a bootstrap sample of the data. SFS random forest trees are constructed with $mtry = 0.5$ for each split, bagging ensemble trees are constructed with $mtry = 1.0$ for each split.

that of trees in the bagging ensemble. Note that the mean of each distribution in the plot, indicated by the vertical line, is equal to the squared bias of the corresponding ensemble as reported in Table 1.

Recall that in §3.1.1, we hint that SFS randomization allows a single decision tree to better capture DGP function $f_2(X_2)$ in the Hidden2D model. In this section, we formally show that SFS randomization reduces the bias of individual trees in a random forests, which reduces model bias compared to bagging.

3.3 An Analysis of How Random Forests Reduce Training Error

Here, we more formally analyze the mechanism behind how random forests reduce training error compared to bagging. Consider a random forest/bagging ensemble that consists of a collection of decision trees, T. Each decision tree is grown to full depth on a bootstrap sample of the original data, which means that each leaf node of the tree contains just a single observation[3]. We can express the prediction of the ensemble for a single observation $X^{(i)}$ by:

$$\hat{f}(X^{(i)}) = \frac{1}{|T|} \sum_{t \in T} L_t(X^{(i)}), \tag{7}$$

where $L_t(X^{(i)})$ denotes the value of the leaf node in tree t that observation $X^{(i)}$ is routed to. Since each decision tree is grown to full depth, the value of the leaf node corresponds to the response of the single observation in the bootstrap sample that the leaf node contains. Functions $\hat{f}(X^{(i)})$ and $L_t(X^{(i)})$ operate element-wise on a single row (observation) in data matrix X.

3. We reiterate here that trees in a random forest are fit on bootstrap samples of the data and not on the original data. Each decision tree perfectly fits the bootstrap sample but may have non-zero training error on the original data.

Let Ψ_i represent the set of decision trees that are fit on bootstrap samples of the data that contain $X^{(i)}$:

$$\Psi_i = \{\text{Trees fit on bootstrap samples that } \textbf{do} \text{ contain } X^{(i)}\},$$

and let Ω_i represent the set of decision trees that are fit on bootstrap samples of the data that do not contain $X^{(i)}$:

$$\Omega_i = \{\text{Trees fit on bootstrap samples that } \textbf{do not} \text{ contain } X^{(i)}\}.$$

We have that:

$$|T| = |\Psi_i| + |\Omega_i|,$$

and that $|\Psi_i| \approx 0.632|T|$. This follows from properties of the bootstrap when the number of bootstrap samples is large.

Tree $t \in \Psi_i$ is fit on a bootstrap sample that contains $X^{(i)}$ so $L_t(X^{(i)}) = Y_i$ since the observation is routed to its own leaf node. We can now decompose the prediction of the random forest/bagging ensemble for observation $X^{(i)}$ into:

$$\hat{f}(X^{(i)}) = \frac{|\Psi_i|}{|T|} Y_i + \frac{1}{|T|} \sum_{t \in \Omega_i} L_t(X^{(i)}) \approx 0.632 \cdot Y_i + \frac{1}{|T|} \sum_{t \in \Omega_i} L_t(X^{(i)}), \tag{8}$$

where the first term is the observed response Y_i times a constant.

The second term of expression (8) is interesting. Consider $L_t(X^{(i)})$ for $t \in \Omega_i$. Tree t is fit on a bootstrapped sample X' of X that **does not** contain observation $X^{(i)}$. As such, observation $X^{(i)}$ is routed to a leaf node in tree t that contains just a single observation $X^{(v)} \in X'$ of the bootstrapped sample, and $L_t(X^{(i)}) = Y_v$.

Voting Points: We refer to $X^{(v)}$ as a voting point for observation $X^{(i)}$, and, as an aside, we briefly discuss why we use this terminology. Voting points in random forests originate from Lin and Jeon (2006) where the authors analyze random forests as a potential nearest neighbor algorithm. The authors define that point X_1 is k-potential nearest neighbor (k-PNN) to point X_2 on dataset X if there exists no other point in dataset X other than X_1 that lies in the hyper-rectangle defined by X_1 and X_2.

Proposition 1 *Voting point $X^{(v)}$ is 1-PNN to observation $X^{(i)}$ in bootstrapped sample X'.*

This proposition follows directly from Proposition 1 in Lin and Jeon (2006). Lin and Jeon (2006) define voting points to be data points that are k-PNN to an observation of interest and that affect the prediction of that observation. Here, $X^{(v)}$ is 1-PNN to $X^{(i)}$ and it is apparent from display (8) that $X^{(v)}$ affects the prediction of $X^{(i)}$.

We return to our training error analysis. Let the function:

$$\mu_{\Omega_i}(X^{(i)}) = \frac{1}{|\Omega_i|} \sum_{t \in \Omega_i} L_t(X^{(i)}) \tag{9}$$

represent the average of all of the voting points for observation $X^{(i)}$. Note that when $|T|$ is large the number of voting points per observation is close to $0.368|T|$; this again follows from properties of the bootstrap. We explicitly characterize the training error of bagging ensembles/random forests in terms of this function in the result below.

Proposition 2 *Given definitions discussed above, the training mean squared error of a random forest and a bagging ensemble can be given by:*

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{f}(X^{(i)}))^2 = \frac{1}{n}\left(\frac{|\Omega_i|}{|T|}\right)^2 \sum_{i=1}^{n}\left(Y_i - \mu_{\Omega_i}(X^{(i)})\right)^2 \approx 0.135\frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \mu_{\Omega_i}(X^{(i)})\right)^2 \tag{10}$$

We derive this expression in Section B.1 of the appendix. This result shows that for random forests/bagging ensembles, the in-sample error of the model depends on $\mu_{\Omega_i}(X^{(i)})$, i.e., the average response of the voting points for each observation of interest $X^{(i)}$. Below, we show how voting points differ in random forests compared to bagging in order to illustrate how random forests reduce training error.

3.3.1 Voting Points of Random Forests vs. Bagging:

We return to our Hidden2D example and consider the simulation study discussed in §3.1.2. Consider the rightmost column of plots in Figure 6; SFS randomization reduces the training error of random forests compared to bagging. To illustrate the effect of voting points, we select the observation in the plot where the difference in training error between bagging and random forests is the largest. We can think of this as the observation where random forests improve the most over bagging; this is our observation of interest.

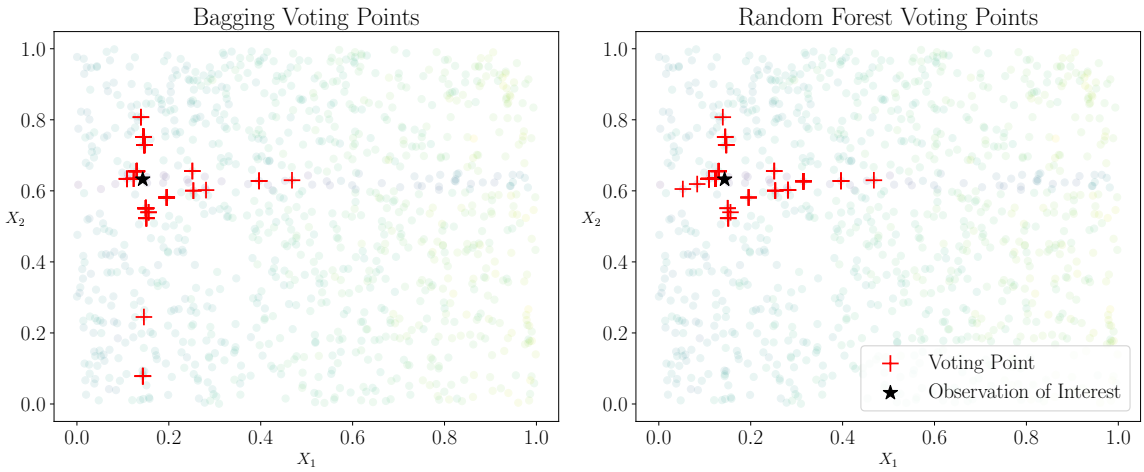


Figure 11: Voting points for our observation of interest in bagging and random forests. The prediction for our observation of interest is a weighted average of these marked points.

In Figure 11, we show the voting points of this observation in both bagging and random forests. Each scatter plot shows the training data, each point corresponds to an observation,

and the points are color-coded with respect to response Y. The black star indicates our observation of interest and each red cross marks to location of a voting point.

From this figure we observe that for the bagging ensemble, there are several voting points that are very far from our observation of interest along the vertical axis, which shows X_2. Recall that in the Hidden2D model, the underlying DGP function $f_2(X_2) = -\mathbb{1}(0.6 \leq X_2 \leq 0.65)$ is only active along the narrow band in X_2 between 0.6 and 0.65. Our observation of interest lies within that region, so we should expect voting points to be clustered along X_2 to reduce training error. As we can see from the right plot in Figure 11, the voting points of the random forests are much more clustered along the vertical axis. This is due to the fact that trees in the random forest tend to split on X_2 earlier and more frequently compared to bagging. Voting points must belong to the same leaf node as the observation of interest, so there cannot be a split between the two points.

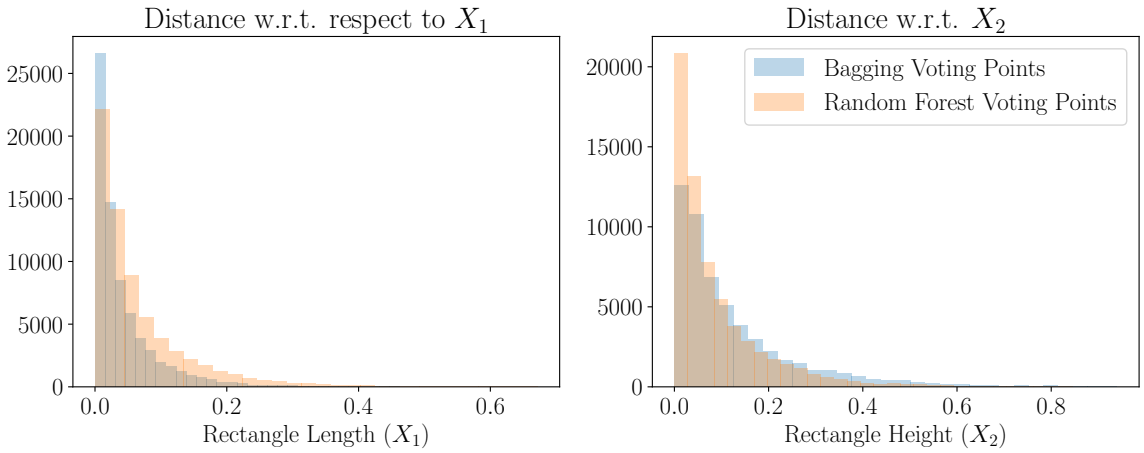


Figure 12: Distribution of distances from observation to voting points for all observations in the data.

We use this procedure to visualize the difference in voting points between random forests and bagging ensembles across all observations in the data, rather than just a single observation of interest. For each observation $X^{(i)} \in X$, we find all of the voting points and report the height and length of the rectangle defined by $X^{(i)}$ and each voting point. The length of the rectangle is the distance between $X^{(i)}$ and the voting point with respect to feature X_1 and the height of the rectangle is the distance between $X^{(i)}$ and the voting point with respect to feature X_2. We plot the distribution of all these lengths and heights, for all observations and voting points, in Figure 12. From this figure, we can see that compared to bagging, random forest voting points are closer to all observations with respect to X_2 and slightly further to all observations with respect to X_1. As such, these voting points better capture the DGP function $f_2(X_2)$ which reduces the training error of a random forest compared to bagging.

4. Generalizing Hidden Patterns

Up until now we focus on a single example of a hidden pattern, the DGP function $f_2(X_2) = -\mathbb{1}(0.6 \leq X_2 \leq 0.65)$ in our Hidden2D model. Informally, we say that the relationship

between feature X_2 and Y, $f_2(X_2)$, is masked by feature X_1 from a bagging ensemble. Trees in a bagging ensemble tend to split on X_1 before X_2 and it is harder for splits to capture $f_2(X_2)$ when conditioned on the preceding splits on X_1. Randomizing the features used in each split allows an ensemble to better capture this hidden pattern and we show in our experiments that this reduces the training error and expected bias of the model. We now formalize this notion of hidden patterns using the definition below.

Definition 1 *Hidden Pattern*

Consider data matrix $X \in \mathbb{R}^{n \times p}$ and let H be a subset of the p features in X, i.e., $H \subset \{1, \ldots, p\}$. Let set H^c denote the complement of set H, let M be a subset of H^c, $M \subset H^c$, and let M^c denote the complement of set M. Matrices X_H and X_{H^c} correspond to sub-matrices of X comprising of columns indexed by H and H^c respectively. Assume that response $Y \in \mathbb{R}^n$ is generated from the model:

$$Y = f_{H^c}(X_{H^c}) + f_H(X_H) + \epsilon, \tag{11}$$

where ϵ is a noise vector, and let estimator g be a bagging ensemble with $mtry = 1.0$.

We say that features H are hidden from bagging ensemble by masking features M if:

$$\big(E[f_H(X_H) - \hat{g}(X)]\big)^2 \geq \big(E[f_H(X_H) - \hat{g}(X_{M^c})]\big)^2, \tag{12}$$

where $\hat{g}(X)$ and $\hat{g}(X_{M^c})$ are the predictions of bagging ensembles fit to predict Y using the data matrices X and X_{M^c} respectively and the expectations are taken with respect to randomness in the training data and the bootstrap in the bagging algorithm. Function $f_H(X_H)$ is a hidden pattern that applies to hidden features H.

Furthermore, we say that a random forest helps capture hidden pattern $f_H(X_H)$ if:

$$\big(E[f_H(X_H) - \hat{g}(X)]\big)^2 \geq \big(E[f_H(X_H) - \hat{r}(X)]\big)^2, \tag{13}$$

where $\hat{r}(X)$ is the prediction of a random forest fit to predict Y using data matrix X, with $mtry < 1.0$, and the expectations are taken with respect to randomness in the training data, the bootstrap in the bagging algorithm, and the bootstrap and mtry randomization in the random forest algorithm.

In our Hidden2D ($n = 1000$, SNR $= 6$) example, we have that $f_H(X_H) = f_2(X_2) = -\mathbb{1}(0.6 \leq X_2 \leq 0.65)$ and that $M = \{1\}$. We repeat the Monte Carlo procedure from Domingos (1997), and which we used for our bias variance decomposition in §3.1.4, for 500 trials to estimate the following quantities.

	$\big(E[f_H(X_H) - \hat{g}(X)]\big)^2$	$\big(E[f_H(X_H) - \hat{g}(X_{M^c})]\big)^2$	$\big(E[f_H(X_H) - \hat{r}(X)]\big)^2$
Hidden2D	0.3326	0.2492	0.3263

From this table, we see that $f_2(X_2)$ satisfies our definition of a hidden pattern and that SFS randomization helps random forests capture the hidden pattern. Recall from Table 1 in §3.1.4 that this corresponds to a decrease in ensemble bias. We note here that our definition of hidden patterns depend on $\big(E[f_H(X_H) - \hat{g}(X)]\big)^2$. Estimating this quantity requires knowledge of the underlying DGP, $f(X)$, and hidden pattern $f_H(X_H)$, functions that are

often unknown in real-world settings. As such, we also focus on the following characteristic of hidden patterns that we observe in practice.

Visualizing Hidden Pattern Characteristics: Consider again the top plot in the leftmost column of Figure 6, in §3.1.2. From this plot, we observe that the training errors of the bagging ensemble fit on the Hidden2D example are higher around the horizontal band $0.6 \leq X_2 \leq 0.65$. We say that the hidden pattern is *active* for data points in this band, since the hidden pattern $f_2(X_2)$ has a large impact on response Y. In fact, for any data point $\{X_1^{(i)}, X_2^{(i)}\} \in X$ such that $0.6 \leq X_2^{(i)} \leq 0.65$ we have that $|f_2(X_2^{(i)})| \geq |f_1(X_1^{(i)})|$. In the bottom plot in the leftmost column of Figure 6, we see that SFS random forests decrease the training error of data points around the horizontal band.

We observe that hidden patterns have this observable characteristic: bagging ensembles have higher training errors on data points where the hidden pattern is active and random forests reduce the training errors of these points. In the sections below, we show that this characteristic consistently appears across several different examples of hidden patterns.

4.1 Examples of Hidden Patterns

In this section, we extend our concept of hidden patterns beyond the Hidden2D example. We use various functions, such as Gaussian, sine, and box functions to generate $f_H(X_H)$ across various dimensions of X_H. Empirically, we show that these hidden patterns satisfy the definition presented above and we visualize their observable characteristics. These examples establish a more general notion of hidden patterns and suggest that hidden patterns may be present in real-world data, an idea we further explore in §4.3.

4.1.1 Hidden Gaussian Spikes

Consider the model:

$$Y = X_1 - \underbrace{2\exp\left(-\frac{(X_2 - 0.28)^2}{2(0.005)^2}\right) + 2\exp\left(-\frac{(X_2 - 0.3)^2}{2(0.005)^2}\right)}_{f_H(X_H)} + \epsilon. \tag{14}$$

The hidden pattern in this model consists of two Gaussian spikes in X_2, labeled by the under-brace in the above display, and masking feature $M = \{1\}$. Using this model we generate $n = 2500$ data points, by sampling X from a $U(0, 1)$ distribution, and set ϵ such that the SNR is equal to 6. We show the hidden pattern by plotting Y against X_2 in the leftmost plot in Figure 13.

On this example, we fit bagging ensembles and SFS random forests ($mtry = 0.5$) with 500 trees and repeat our Monte Carlo procedure (§3.1.4) across 500 trials to estimate the following quantities.

	$\left(E[f_H(X_H) - \hat{g}(X)]\right)^2$	$\left(E[f_H(X_H) - \hat{g}(X_{M^c})]\right)^2$	$\left(E[f_H(X_H) - \hat{r}(X)]\right)^2$
HiddenSpike	0.3337	0.2497	0.3229

We note that $f_H(X_H)$ satisfies our definition of a hidden pattern and that randomization helps random forests capture the pattern. We also apply the same Monte Carlo procedure

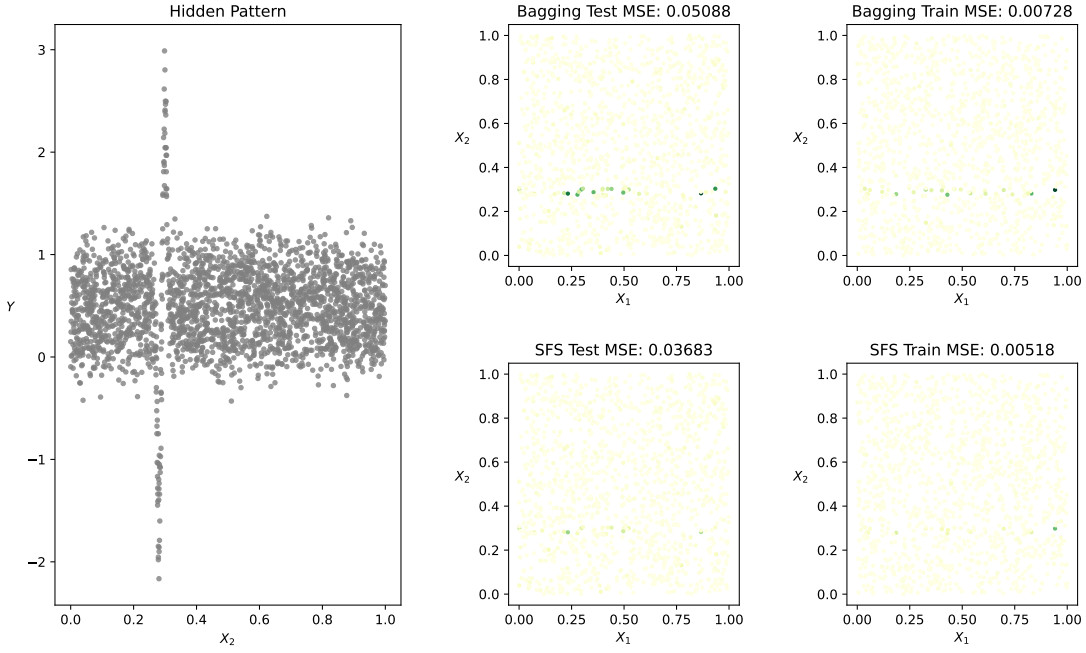


Figure 13: Visualization of a Gaussian spike hidden pattern. The four plots on the right side of this plot show the characteristic of a hidden pattern, that bagging ensembles have higher errors in regions where the hidden pattern is active compared to SFS random forests.

to estimate the bias and variance of a bagging ensemble and a SFS random forest on this example. From the table below, we observe that SFS random forests again reduce ensemble bias and variance compared to bagging.

HiddenSpike	Bias^2	Variance
SFS Random Forest	0.00310	0.0058
Bagging Ensemble	0.00821	0.0107

Finally, the four plots on the right in Figure 13 show the training and test errors of a bagging ensemble and a SFS random forest for each data point in the example, using a 50-50 train-test split. From the top right plot we see that training error for bagging ensembles is high for the data-points affected by the hidden pattern, in the region around $X_2 \approx 0.3$. SFS random forests reduces the training error of these points, as expected.

4.1.2 Hidden Sinusoidal

We repeat the analyses discussed in the section above on the following model:

$$Y = X_1 - \underbrace{\left(3\sin(50\pi X_2)\mathbb{1}_{X_2\in[0.36,0.4]} + 3\sin(70\pi X_2)\mathbb{1}_{X_2\in[0.6,0.63]} + 3\sin(90\pi X_2)\mathbb{1}_{X_2\in[0.8,0.82]}\right)}_{f_H(X_H)} + \epsilon, \tag{15}$$

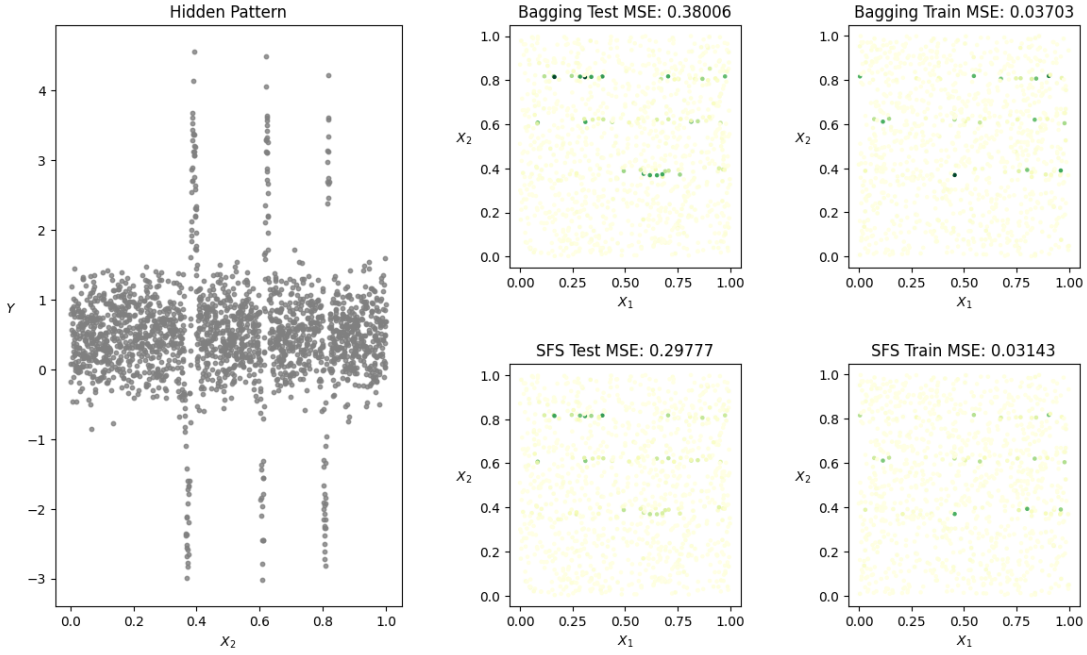


Figure 14: Visualization of the sine hidden pattern.

where the hidden pattern $f_H(X_H)$ consists of 3 sinusoidal patterns defined along intervals of X_2 and masking feature $M = \{1\}$. We generate $n = 2000$ data points using this model, with ϵ set so that SNR $= 6$; the leftmost plot in Figure 14 shows a visualization of the hidden pattern.

We fit bagging ensembles and SFS random forests ($mtry = 0.5$) and repeat the procedure above to estimate the following quantities.

	$\big(E[f_H(X_H) - \hat{g}(X)]\big)^2$	$\big(E[f_H(X_H) - \hat{g}(X_{M^c})]\big)^2$	$\big(E[f_H(X_H) - \hat{r}(X)]\big)^2$
HiddenSine	0.42412	0.24951	0.40307

Again, we see that $f_H(X_H)$ satisfies our definition of a hidden pattern and furthermore we show in the table below that SFS random forests reduce both bias and variance compared to bagging in this example.

HiddenSine	Bias^2	Variance
SFS Random Forest	0.06786	0.03495
Bagging Ensemble	0.0876	0.05338

Finally, the four plots on the right of Figure 14 show the training and test errors of each ensemble for each data point. Again, we observe that our HiddenSine model shows the characteristics of a hidden pattern; the training errors of the bagging ensemble are concentrated around the points impacted by the hidden pattern, the three horizontal bands, and these errors are reduced by the random forest.

4.1.3 Hidden Square and Hidden Circle

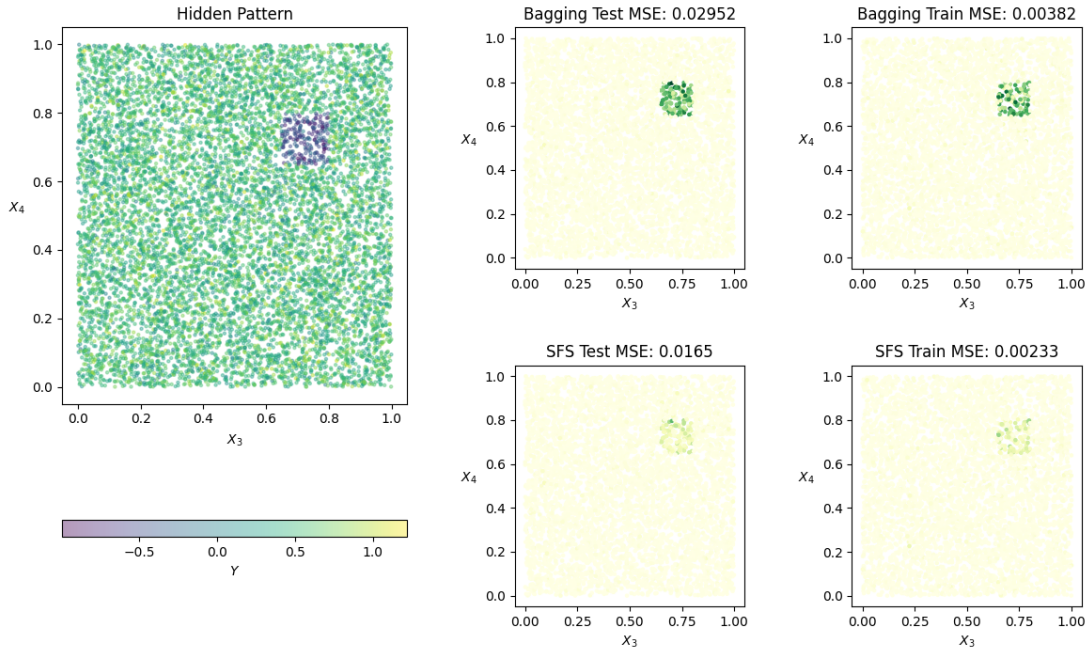


Figure 15: HiddenSquare visualization

Now consider the model:

$$Y = 0.5X_1 + 0.5X_2 - \underbrace{\left(\mathbb{1}_{X_3 \in [0.65,0.8], X_4 \in [0.65,0.8]}\right)}_{f_H(X_H)} + \epsilon, \tag{16}$$

which we call HiddenSquare. The hidden pattern $f_H(X_H)$ is defined by a box in X_3 and X_4 and we have masking features $M = \{1, 2\}$; the hidden pattern is jointly a function of these features. Using this HiddenSquare model we generate $n = 10000$ data points with ϵ set such that the SNR of the data is 6. In the left plot of Figure 15, we show a visualization of this hidden pattern by plotting X_3 and X_4; the color of points in this scatter plot show Y.

Now, we fit bagging ensembles and SFS random forests, with $mtry = 0.333$, and repeat the procedures discussed above to estimate the following quantities.

	$\left(E[f_H(X_H) - \hat{g}(X)]\right)^2$	$\left(E[f_H(X_H) - \hat{g}(X_{M^c})]\right)^2$	$\left(E[f_H(X_H) - \hat{r}(X)]\right)^2$
HiddenSquare	0.30046	0.25046	0.28862

HiddenSquare	Bias^2	Variance
SFS Random Forest	0.00329	0.000871
Bagging Ensemble	0.00991	0.002439

From these results, we see that $f_H(X_H)$ in our HiddenSquare models satisfies our definition of a hidden pattern and that SFS random forests help capture the pattern. Moreover, SFS random forests again reduce both the bias and variance of the ensemble compared to bagging.

In the four right plots in Figure 15, we show the errors of each ensemble on each data point in the HiddenSquare model. We plot these errors against X_3 and X_4 in these scatter plots and show again that the training errors of the bagging ensemble are high for data points impacted by the hidden pattern. These points correspond to the box defined on X_3 and X_4 and SFS random forests reduce the training error in this region.

We repeat this entire analysis on a similar model, defined by:

$$Y = 0.74X_1 + 0.26X_2 - 0.1X_3^2 - \underbrace{\left(\mathbb{1}_{(X_4-0.3)^2+(X_5-0.3)^2 \leq 0.0075}\right)}_{f_H(X_H)} + \epsilon, \tag{17}$$

which we call HiddenCircle. The hidden pattern $f_H(X_H)$ is now defined by a circle in X_4 and X_5 and masking features $M = \{1, 2, 3\}$. Using this model, we again generate $n = 10000$ data points with SNR $= 6$ and show a visualization of the hidden pattern in the left plot in Figure 16.

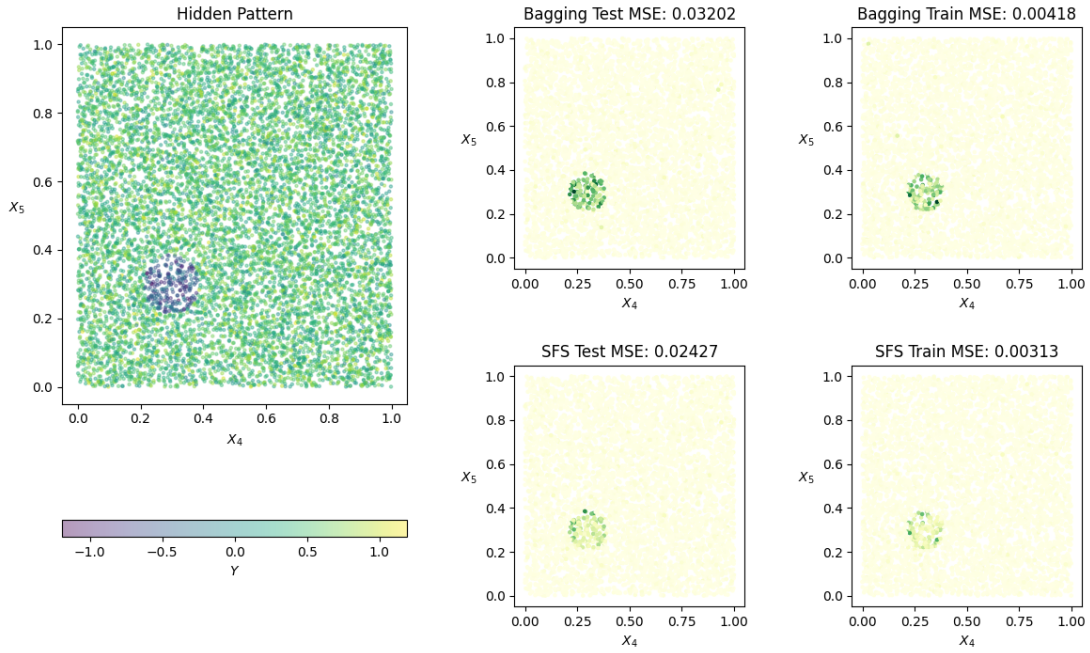


Figure 16: HiddenCircle visualization.

Repeating the analysis from the HiddenSquare example, we fit random forests ($mtry = 0.333$) and bagging ensembles and estimate our quantites of interest. As in the HiddenSquare

	$\left(E[f_H(X_H) - \hat{g}(X)]\right)^2$	$\left(E[f_H(X_H) - \hat{g}(X_{M^c})]\right)^2$	$\left(E[f_H(X_H) - \hat{r}(X)]\right)^2$
HiddenCircle	0.28233	0.21912	0.26828

HiddenCircle	Bias^2	Variance
SFS Random Forest	0.00798	0.000875
Bagging Ensemble	0.01208	0.002378

example, we see that our HiddenCircle hidden pattern satisfies our definition of hidden patterns, and that SFS random forests reduce both bias and variance. From the plots in Figure 16, we again see that the errors of the bagging ensemble are concentrated around points impacted by the hidden pattern, and that SFS random forests reduces the error of these points.

4.1.4 Hidden Cube

Finally, we consider the model:

$$Y = 0.5X_1 + 0.5X_2 - \underbrace{2\left(\mathbb{1}_{X_3,X_4,X_5\in[0.1,0.4]}\right)}_{f_H(X_H)} + \epsilon, \tag{18}$$

which we call HiddenCube. Here, the hidden pattern $f_H(X_H)$ is defined by the cube in X_3, X_4, and X_5 and masking features $M = \{1, 2\}$. We use this model to generate $n = 2000$ data points with an SNR of 6. In the leftmost plot in Figure 17, we show a visualization of this hidden pattern by plotting X_3, X_4, and X_5; the color of each point in the plot shows Y.

Repeating the procedures from above, we fit random forests ($mtry = 0.333$) and bagging ensembles and estimate the following quantities. Again, we see that $f_H(X_H)$ satisfies our

	$\left(E[f_H(X_H) - \hat{g}(X)]\right)^2$	$\left(E[f_H(X_H) - \hat{g}(X_{M^c})]\right)^2$	$\left(E[f_H(X_H) - \hat{r}(X)]\right)^2$
HiddenCube	0.4201	0.3698	0.4006

definition of a hidden pattern and that SFS random forests help better capture this pattern. As such, SFS random forests have lower bias and variance compared to bagging ensembles when fit on this HiddenCube model.

Finally, the right four plots in Figure 17 show the errors of each ensemble plotted against $\{X_3, X_4, X_5\}$. From this plot, we again see that the training errors of the bagging ensemble are high for points impacted by the hidden pattern, the dark cube in the plots, and the SFS random forests reduces the errors of these points.

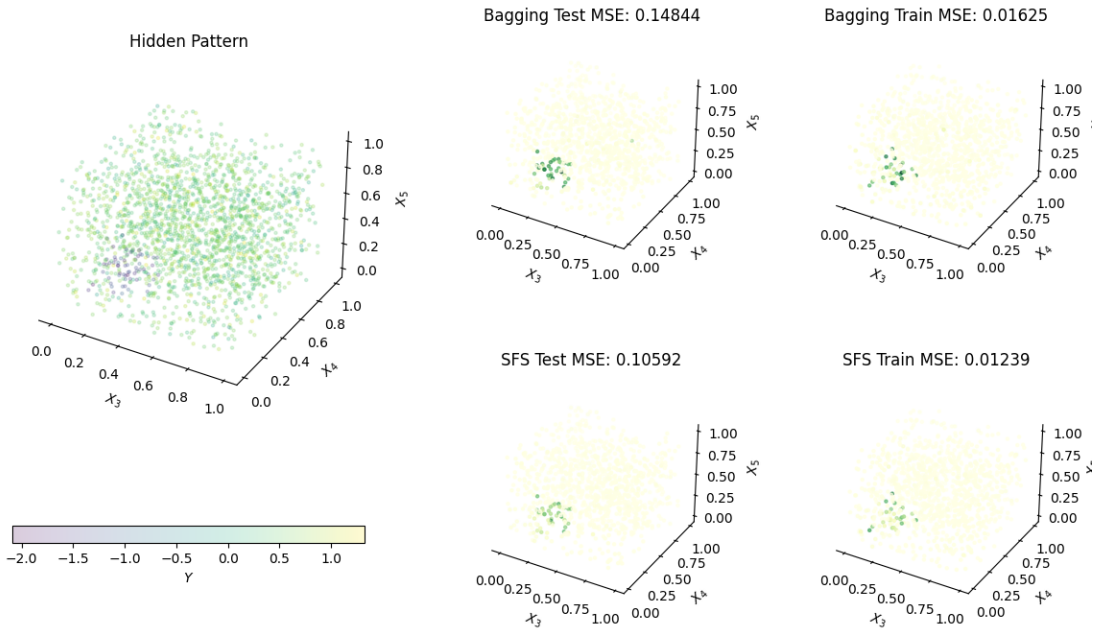


Figure 17: Visualization of cube hidden pattern.

HiddenCube	Bias^2	Variance
SFS Random Forest	0.1200	0.001366
Bagging Ensemble	0.1300	0.003410

4.2 Additional Empirical Studies

In the section above, we explore in detail several examples of hidden patterns and discuss their characteristics. Here, we conduct a large scale empirical study to show that random forests reduce both bias and variance across a range of examples that vary in terms of data generating procedures and dataset sizes. We generate models for this study using the formula:

$$Y = \underbrace{f_{H^c}(X_{H^c}) + f_H(X_H)}_{f(X)} + \epsilon, \tag{19}$$

where DGP function $f(X) = f_{H^c}(X_{H^c}) + f_H(X_H)$. For the functions $f_{H^c}(X_{H^c})$ we consider the MARS and MARSadd functions defined in displays (2) and (3) along with the following functions:

$$0.3X_1 - 0.5X_2 + 0.1X_3 + 0.1X_4 + X_5 \tag{Linear}$$

$$2.0 + 3.5X_1 - 1.2X_1^2 + 4.0X_2 + 0.8X_2^3 \tag{Polynomial}$$

$$\sqrt{X_1^2 + \left(X_2X_3 - \frac{1}{X_2X_4 + 0.1}\right)^2} \tag{Friedman2}$$

$$\arctan\left(X_2X_3 - \frac{1}{X_2X_4 + 0.1}\right). \qquad \text{(Friedman3)}$$

For hidden patterns $f_H(X_H)$ we consider functions of the form found in Hidden2D, HiddenSpike, and HiddenSquare.

We consider various problem sizes with $n \in \{1000, 5000, 10000, 20000\}$. Similar to the studies above, we sample data matrix X from a $U(0,1)$ distribution. For all of the models generated, we set ϵ such that the SNR of the data is equal to 6. The full expression for each model considered in our study can be found in Section C.1 of the appendix.

On each model, we repeat the Monte Carlo procedure used in Domingos (2000), and in sections 3.1.4 and 4.1 of our paper, 500 times to estimate the bias and variance of a bagging ensemble ($mtry = 1.0$) and random forest ($mtry < 1.0$) fit on the data. The ensembles are grown to have 500 trees and are identical except for $mtry$.

We show the results of our empirical study in Table 2. From this table, we observe that random forests can reduce both bias and variance compared to bagging ensembles across a wide range of problems. In each of these models for every problem size, the bias and variance of the random forest is lower than that of the bagging ensemble.

Finally, we note that our definition of hidden patterns, along with the examples and experiments discussed above, assumes that the data-generating procedure is an additive combination of the effects of the hidden pattern and the masking features, see display (11). This assumption may not be necessary, and exploring whether (12) and (13) hold under other data-generating procedures is an interesting direction for future research.

Model Name	n	Random Forest Bias^2	Bagging Bias^2	Random Forest Variance	Bagging Variance
Friedman2 Hidden2D	1000	0.4543	0.4642	0.1843	0.2429
Friedman2 Hidden2D	5000	0.3234	0.4113	0.1284	0.1732
Friedman2 Hidden2D	10000	0.2303	0.3341	0.1078	0.1455
Friedman2 Hidden2D	20000	0.181	0.3135	0.0957	0.1294
Friedman3 Hidden2D	1000	0.071	0.0953	0.0052	0.0158
Friedman3 Hidden2D	5000	0.012	0.0231	0.0043	0.0083
Friedman3 Hidden2D	10000	0.0138	0.0276	0.0013	0.0048
Friedman3 Hidden2D	20000	0.0098	0.0223	0.001	0.004
Polynomial Hidden2D	1000	1.3861	1.5128	0.1029	0.2487
Polynomial Hidden2D	5000	0.9514	1.4184	0.0579	0.1646
Polynomial Hidden2D	10000	0.6536	1.1388	0.0442	0.1375
Polynomial Hidden2D	20000	0.4605	0.9383	0.0364	0.1198
Polynomial HiddenSquare	1000	0.6893	0.6923	0.0968	0.1465
Polynomial HiddenSquare	5000	0.4917	0.555	0.0296	0.103
Polynomial HiddenSquare	10000	0.4497	0.6177	0.0255	0.0872
Polynomial HiddenSquare	20000	0.2923	0.5334	0.0204	0.0752
Polynomial HiddenSpike	1000	0.4035	0.4457	0.1026	0.1518
Polynomial HiddenSpike	5000	0.2131	0.4224	0.0359	0.102
Polynomial HiddenSpike	10000	0.1437	0.3254	0.0295	0.0879
Polynomial HiddenSpike	20000	0.0921	0.2315	0.0469	0.0792
Linear Hidden2D	1000	0.0813	0.0953	0.0062	0.0162
Linear Hidden2D	5000	0.0548	0.0867	0.0035	0.0107
Linear Hidden2D	10000	0.044	0.0757	0.0028	0.0094
Linear Hidden2D	20000	0.0334	0.0624	0.0022	0.0083
Linear HiddenSquare	1000	0.1865	0.1928	0.0226	0.037
Linear HiddenSquare	5000	0.0345	0.0377	0.0023	0.0075
Linear HiddenSquare	10000	0.0384	0.0452	0.0019	0.006
Linear HiddenSquare	20000	0.0343	0.0436	0.0016	0.0053
Mars Hidden2D	1000	59.9471	70.2535	5.1018	10.7458
Mars Hidden2D	5000	33.4228	40.5931	2.882	6.9979
Mars Hidden2D	10000	20.2106	24.7232	2.1899	5.5618
Mars Hidden2D	20000	13.431	15.426	3.1332	4.7419
Mars HiddenSquare	1000	63.7704	64.8151	15.1242	26.2896
Mars HiddenSquare	5000	119.0924	124.2567	4.849	17.9954
Mars HiddenSquare	10000	140.8299	154.133	3.8958	14.8447
Mars HiddenSquare	20000	121.5901	142.0494	3.2596	12.7916
Mars HiddenSpike	1000	103.9615	119.2452	13.2049	22.7455
Mars HiddenSpike	5000	32.0277	35.1137	6.9643	13.627
Mars HiddenSpike	10000	12.2086	14.1591	4.4306	7.9408
Mars HiddenSpike	20000	30.7046	39.0582	4.1251	7.4426
Marsadd HiddenSquare	1000	134.3366	143.7545	4.979	15.0782
Marsadd HiddenSquare	5000	31.061	41.9002	4.6501	7.9683
Marsadd HiddenSquare	10000	23.6029	32.295	4.3727	7.7874
Marsadd HiddenSquare	20000	8.8271	15.2515	2.3843	4.7962
Marsadd Hidden2d	1000	4.602	4.9909	0.4208	0.9364
Marsadd Hidden2d	5000	2.1736	2.523	0.2286	0.5734
Marsadd Hidden2d	10000	1.4905	1.737	0.1782	0.4576
Marsadd Hidden2d	20000	1.0318	1.2295	0.1465	0.3671
Marsadd HiddenSpike	1000	19.7094	21.4458	2.8458	4.7252
Marsadd HiddenSpike	5000	1.0263	1.6187	0.0918	0.27
Marsadd HiddenSpike	10000	0.7592	1.3841	0.0734	0.2332
Marsadd HiddenSpike	20000	0.5599	1.1348	0.0616	0.2009

Table 2: Results of our empirical study, cells with the lower bias or variance are highlighted in green. Random forests reduce bias and variance compared to bagging across all of the models and settings considered in the study.

4.3 Case Study: Real World Hidden Pattern

We conclude this section with a case study on what we believe to be a hidden pattern on a real-world dataset: the California Housing Dataset from Pace and Barry (1997). This dataset has 20640 observations, each data point corresponds to a census block group in California, and 8 features: latitude, longitude, median income, population, house age, average rooms per household, average bedrooms per household, and average occupancy per household. Our goal is to predict the median home value of each census block group. We note two interesting details about this problem. First, we expect median home value to be highly associated with median income since individuals with higher incomes are more likely to purchase more expensive homes. Second, we expect certain geographical regions in California, such as the San Francisco Bay Area and the Los Angeles Metropolitan Area to have higher home values.

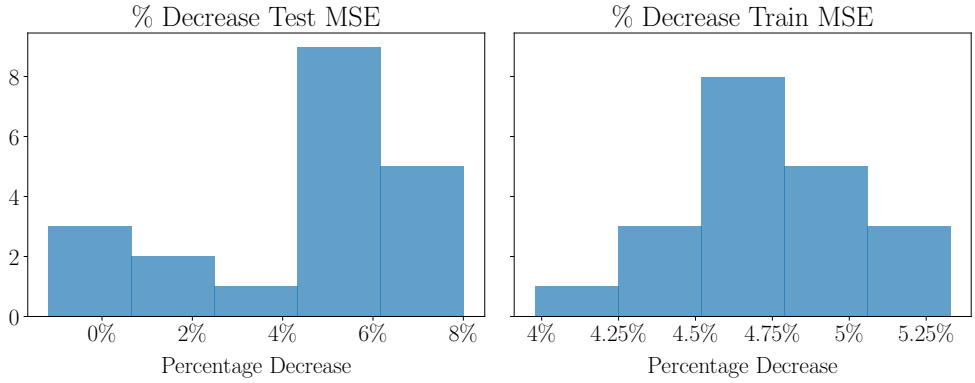


Figure 18: Distributions of percent decreases in training and test MSE for random forests vs. bagging. Random forests consistently decrease training error over bagging in the California Housing example.

We perform a 20-fold cross validation and on each fold we fit a bagging ensemble with $mtry = 1.0$ and a random forest with $mtry = 0.333$ of 2000 full-depth decision trees. We report the percent decrease in training and test mean-squared error between the two ensembles, given by:

$$\frac{\text{MSE(bagging)} - \text{MSE(random forest)}}{\text{MSE(bagging)}} \times 100\%,$$

where percent decreases above 0% indicate that random forests outperform bagging. The distributions of these percent decreases across all folds are shown in Figure 18. We note from the right histogram that random forests consistently reduce the training error of the ensemble over bagging by around 4%. This corresponds to an average decrease in test MSE of around 4.4% as well. From these plots, we observe that our random forest consistently outperforms bagging on this example.

On each fold, we also record the 50 training data points where the difference in squared error between bagging and random forests is the largest. These data points can be viewed as the census block groups where random forests improve the most over bagging. In Figure 19 we show a map of all the census block groups in California, each census block group is

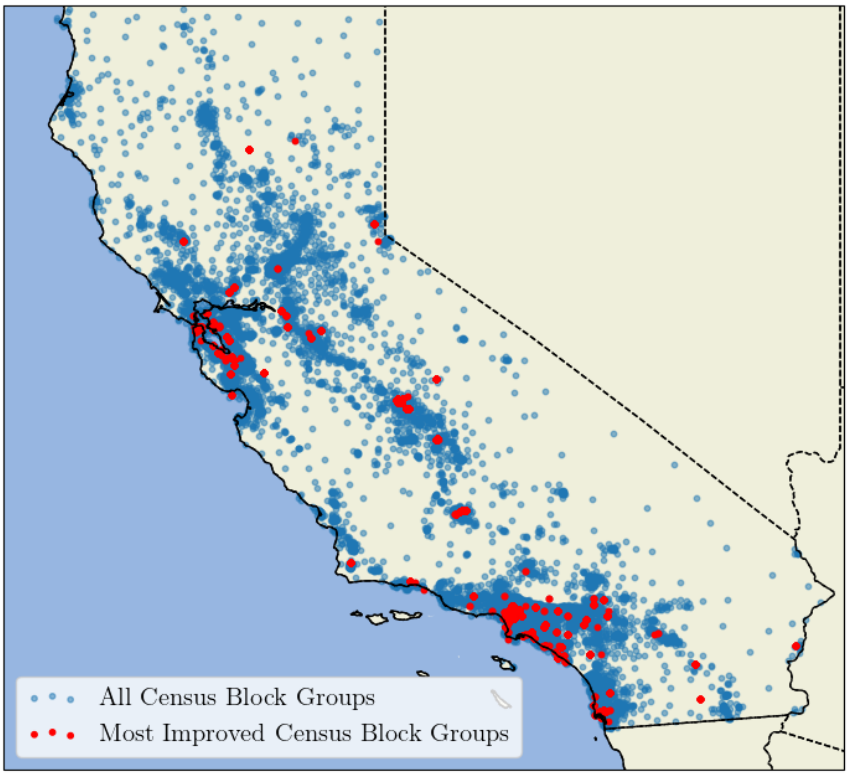


Figure 19: Visualization of census block groups in California. The census block groups where random forests improve the most over bagging are highlighted in red.

shown by a blue point. The red points show the census block groups where random forests reduce training error the most compared to bagging, across all folds. From this plot, we observe that random forests improve over bagging the most in the SF Bay Area and the LA Metropolitan Area.

We hypothesize that the latitude and longitude features in this dataset contain hidden patterns similar to the one in the HiddenSquare model discussed in §4.1.3. Census block groups within the latitude-longitude boundaries of the SF Bay Area or LA Metropolitan Area are likely to have significantly increased median home values compared to surrounding regions. These hidden patterns, however, may be obscured by other influential features such as median income, which has a nearly linear relationship with median home value (appendix D.1). Random forests may help uncover this hidden pattern; this would explain the consistent decrease in training error compared to bagging shown in Figure 18 as well as the geographical distribution in improvements shown in Figure 19.

We test our hypothesis using this sample splitting procedure. Note that in this real-world example, the underlying hidden pattern function $f_H(\text{latitude}, \text{longitude})$ is unknown. To estimate this function, we first randomly take a 50-50 split of the dataset. On the first half of the data, we fit a bagging ensemble of 1000 decision trees to predict housing price using *only* the hidden features in $H = \{\text{latitude}, \text{longitude}\}$. On the second half of the data, we use this bagging ensemble to produce a plug-in estimate for the underlying hidden pattern function:

$\hat{f_H}(X_H)$. We then fit the following ensembles to predict housing price on the second half of the dataset:

- $g(X)$: a bagging ensemble that uses all of the features.
- $g(X_{M^c})$: a bagging ensemble that uses all of the features except median income, our hypothesized masking feature X_M.
- $r(X)$: a random forest with $mtry = 0.333$.

We repeat this data splitting procedure 500 times to estimate the following quantities.

	$\left(E[\hat{f_H}(X_H) - \hat{g}(X)]\right)^2$	$\left(E[\hat{f_H}(X_H) - \hat{g}(X_{M^c})]\right)^2$	$\left(E[\hat{f_H}(X_H) - \hat{r}(X)]\right)^2$
CAHousing	0.240	0.223	0.231

The distributions for these estimates can be found in §D.2 of the appendix. These results support the hypothesis that features latitude and longitude contain a hidden pattern masked by median income and that random forests help capture this feature.

5. Bias Reduction, $mtry$, and Random Forest Tuning

We conclude this paper by analyzing the effect of the $mtry$ parameter on bias reduction in random forests and its implications for random forest tuning. First, we present an empirical study to explore how $mtry$ impacts bias reduction. We consider two models, hMARS:

$$Y = 10sin(\pi X_1 X_2) + 20(X_3 - 0.05)^2 + 10X_4 + 5X_5 - 30 \times \mathbb{1}(0.6 \leq X_6 \leq 0.65) \\ -35 \times \mathbb{1}(0.55 \leq X_7 \leq 0.6) + \epsilon, \tag{20}$$

and hMARSadd:

$$Y = 0.1e^{4X_1} + \frac{4}{1 + e^{-20(X_2 - 0.5)}} + 3X_3 + 2X_4 + X_5 - 10 \times \mathbb{1}(0.6 \leq X_6 \leq 0.65) \\ -7.5 \times \mathbb{1}(0.55 \leq X_7 \leq 0.6) + \epsilon, \tag{21}$$

where each model contains hidden patterns in features X_6 and X_7 that are similar to the ones found in our Hidden2d model. Using each model, we generate $n = 1000$ observations by sampling X from a $U(0, 1)$ distribution, with ϵ set such that the SNR $= 6$. We repeat 500 trials of the Monte Carlo estimation procedure used in §4.2 while varying $mtry \in \{0.1, 0.2, \ldots, 0.9, 1.0\}$ to estimate the bias and variance of bagging ensembles ($mtry = 1.0$) and random forests ($mtry < 1.0$).

Figure 20 shows the results of this study. We see that on the hMARS and hMARSadd models, increasing randomization by reducing $mtry$ from 1.0 to 0.333 reduces both bias and variance. However, we also observe that setting $mtry$ to the smallest value drastically increases bias. In this case, a single randomly selected feature is considered for each split and the resulting ensemble has high bias. We see here that selecting an appropriate value for $mtry$ is crucial for reducing bias in a random forest.

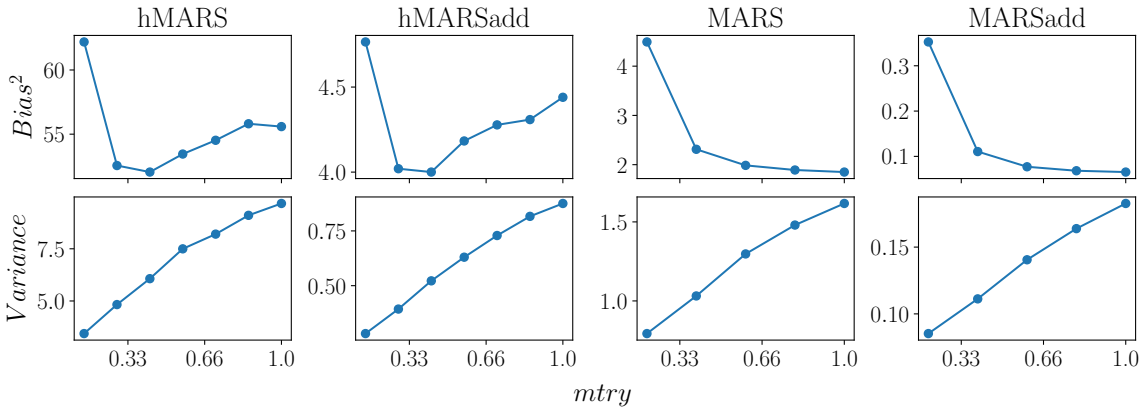


Figure 20: Bias-variance decomposition for random forests and bagging ensembles while varying $mtry$; the right-most points show bagging ensembles with $mtry = 1.0$.

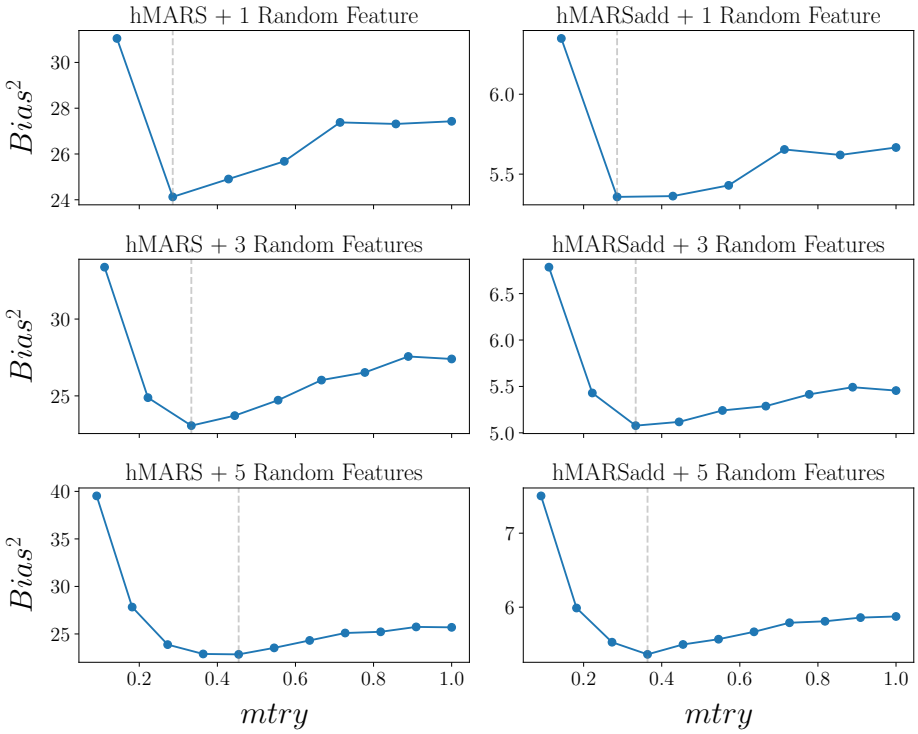


Figure 21: Adding non-informative features to the hMARS and hMARSadd datasets increases the best value of $mtry$ for bias reduction.

5.1 Non-informative Features and $mtry$

In this section, we empirically demonstrate that selecting the best value of $mtry$ for bias reduction also depends on the number of non-informative features in the data.

Consider the case where we add 1, 3, and 5 non-informative (randomly generated) features to the hMARS and hMARSadd dataset (with 1000 datapoints and SNR = 6) and repeat

the bias-variance analysis conducted above. We show in Figure 21 the bias term (vertical axes) plotted against *mtry* (horizontal axes) for each dataset[4]. The vertical dashed gray line in each panel shows the value of *mtry* that minimizes the bias term. From this figure, we observe that as the number of non-informative random features increases, the value of *mtry* that corresponds to the smallest bias increases as well. We explore why this occurs below.

In this simulation study, we empirically demonstrate that *mtry* controls the trade-off between the ability of a random forest to capture hidden patterns and its sensitivity to non-informative features. As such, it may be beneficial in certain scenarios to increase *mtry* on datasets with many non-informative features. Each potential split in a random forest considers a random subset of $\lceil mtry \times p \rceil$ features. Consider the case where in this subset, every single feature is non-informative. The resulting split would be essentially random, i.e., a *non-informative* split, which may increase bias.

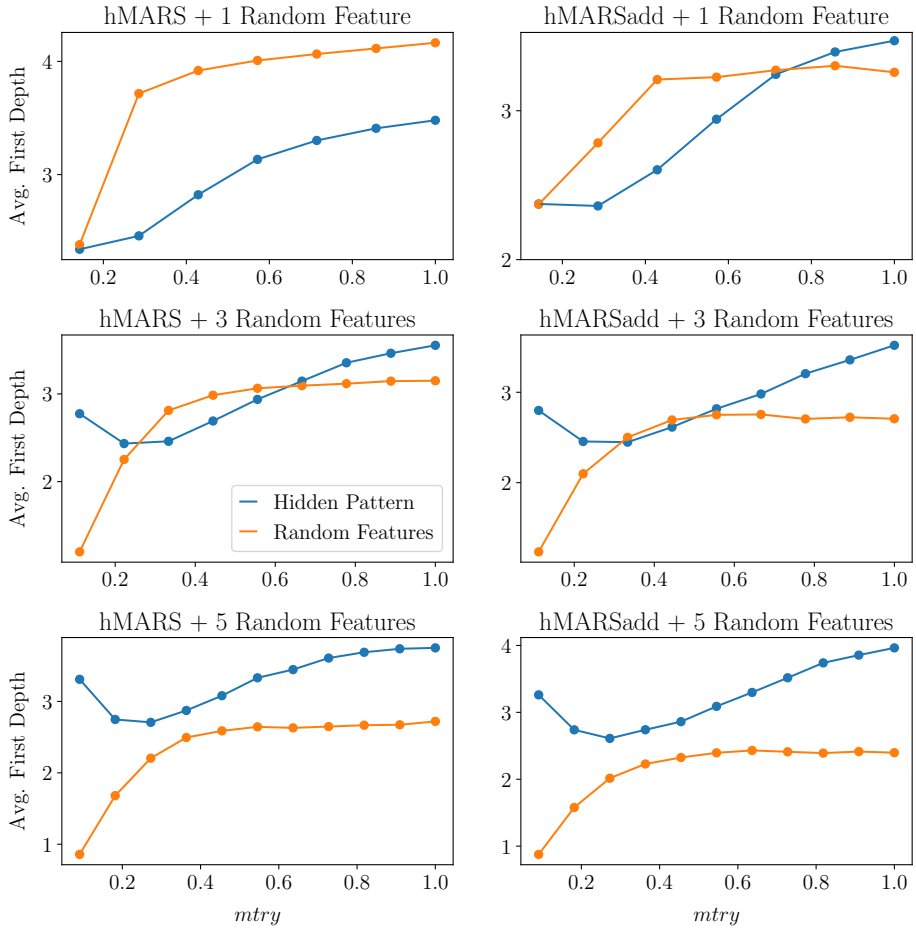


Figure 22: Feature importance measured via average first depth for hMARS and hMARSadd datasets (1000 datapoints and SNR = 6) with added random features. Lower values of average first depth means that the feature is more influential to the model.

4. We show plots of the variance terms in the Appendix E.

We use this procedure to assess how the influence of *non-informative* splits, splits that only consider non-informative features, varies with *mtry*. For each feature, we examine in each tree the depth of the first split that uses the feature; this can be thought of as a proxy for how influential the feature is (Breiman, 2017). We average across all trees in the ensemble to obtain the average first depth for each feature; features with a lower average first depth should be more influential. Following the procedure discussed above, we assess how the average first depth of non-informative features and features with hidden patterns vary with *mtry* when we fit random forests on the hMARS and hMARSadd models with $n = 1000$ and SNR $= 6$. The results are shown in Figure 22.

In this figure, the horizontal axes show *mtry* and the vertical axes show the average first depth for the non-informative features (orange) and the features with hidden patterns (blue). We observe that in all panels, decreasing *mtry* beyond a certain point drastically decreases the average first depth of non-informative features, which corresponds to a large increase in the influence of non-informative splits. We also observe that decreasing *mtry* up to a certain point decreases the average first depth of hidden pattern features, which corresponds to an increase in the influence of features with hidden pattern. Here, *mtry* balances the influence of features with hidden patterns with the influence of non-informative features. Increasing the influence of features with hidden patterns can reduce bias, but increasing the influence of non-informative features can increase bias. It is important to mention that adding random features to a dataset and inducing non-informative splits in a random forest reduces the effective DoF of the ensemble, a key insight highlighted by Mentch and Zhou (2022). The authors propose augmented bagging, where non-informative features are injected into the dataset, and show that the procedure can improve the predictive performance of bagging ensembles in low SNR settings. In low SNR settings, increasing the influence of non-informative features may in fact improve the predictive performance of the model by reducing variance.

In many real-world datasets the underlying data generating procedure, as well as the SNR of the data, the presence of hidden patterns, and the number of non-informative features, are unknown. Consequently, it may be important to tune *mtry*, as well as other parameters in the random forest, to improve performance. In the section below, we explore the effects of tuning *mtry*, and other parameters, on the predictive performance of random forests on real world data.

5.2 Exploring Random Forest Tuning

Throughout this paper, we mention several parameters in a random forest that can improve predictive performance through different mechanisms. We focus on the *mtry* parameter, and show that *mtry* can reduce both bias and variance. In §2.2, we also discuss the TRIM procedure which reduces the effective DoF and variance of an ensemble by controlling tree size. Zhou and Mentch (2023) further explores this idea, and the authors show that tuning tree depth can improve predictive performance in low SNR settings. Finally, in the section above, we mention the augmented bagging procedure introduced in Mentch and Zhou (2022). Adding non-informative noise features to a dataset can also regularize bagging ensembles when SNR is low.

Dataset Name	Observations	Features	R^2 Bagging	Best Tuning Method
autoMpg	398	25	0.818	*mtry*
no2	500	7	0.602	*mtry*
boston	506	22	0.877	tree depth
stock	950	9	0.985	*mtry*
socmob	1156	39	0.804	*mtry*
spacega	3107	6	0.662	tree depth
abalone	4177	10	0.546	tree depth
winequality	6497	11	0.528	*mtry*
wind	6574	14	0.783	tree depth
puma32H	8192	32	0.933	*mtry*
bank32nh	8192	32	0.513	*mtry*
cpusmall	8192	12	0.977	*mtry*
kin8nm	8192	8	0.718	*mtry*
pol	15000	48	0.988	*mtry*
elevators	16599	18	0.839	*mtry*
houses	20640	8	0.821	*mtry*
house16H	22784	16	0.635	*mtry*
mv	40768	14	0.982	*mtry*

Table 3: List of datasets used in our parameter tuning experiment. The datasets considered span a range of dimensions and SNRs. The rightmost column shows the tuning method that works the best on each dataset.

In this section, we compare the impact of tuning these parameters on real-world data, following the experimental procedure discussed below. We use 17 regression datasets from OpenML and use 5-fold CV on each dataset; dataset names and dimensions are shown in the 3 leftmost columns of Table 3. We split each training fold 70-30 into a training set and validation set and fit bagging ensembles with 1000 decision trees, while tuning the following methods.

- We tune **mtry** across the following 12 values: $\{0.1, 0.18, 0.26, 0.35, 0.43, 0.51, 0.59, 0.67, 0.75, 0.8, 0.86, 0.92\}$.

- We tune the minimum number of samples per leaf node as a proxy for tree depth; this procedure was used in Zhou and Mentch (2023). We vary **node size** across the following 12 values: $\{1, 3, 5, 0.06n, 0.11n, 0.17n, 0.22n, 0.28n, 0.33n, 0.39n, 0.44n, 0.5n\}$, where n is the number of observations in the dataset.

- We tune the number of additional **noise features** q and the **correlation** r of the features added in augmented bagging, following the procedure discussed in Mentch and Zhou (2022). We vary $q \in \{p/2,\ p,\ 3p/2,\ 2p\}$, where p is the number of features in the dataset, and $r \in \{0,\ 0.4,\ 0.9\}$, where r is the correlation between each noise feature and a feature in the dataset. This corresponds to a grid of 12 settings for q and p.

For each method, we select the parameters that yield the minimum validation error, refit the model on the entire training set, and evaluate the R^2 of the model on the test set.

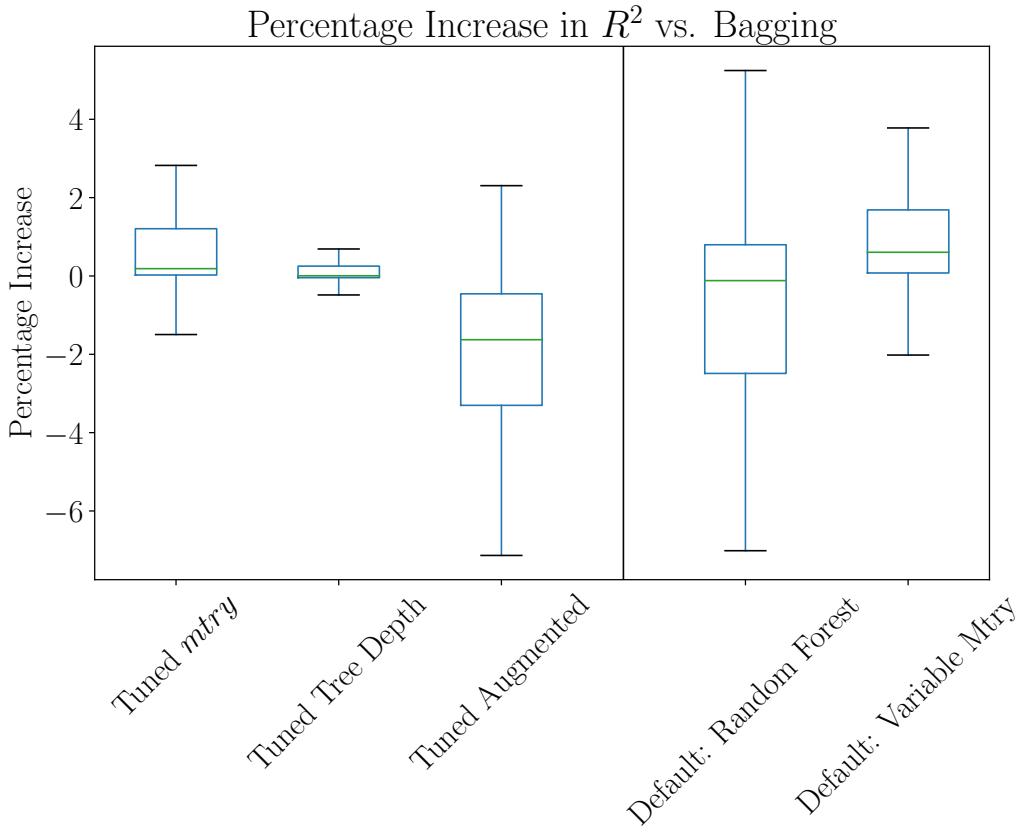


Figure 23: Comparison of various tuning methods against bagging. The vertical axis shows the percentage increase in R^2, values above 0 indicate the method outperforms bagging.

Across all datasets and folds in our experiment, we report the percent increase in test R^2 between the best model found by each tuning method and a standard bagging ensemble, of 1000 decision trees grown to full depth with $mtry = 1.0$. This is given by the expression:

$$\frac{\text{Tuned Method } R^2 - \text{Bagging } R^2}{\text{Tuned Method } R^2}.$$

We show the distribution of our results in the left three boxplots of Figure 23. The vertical axis shows the percentage increase in R^2 for each method compared to bagging; higher values indicate greater improvements of the tuned methods over bagging.

From this plot, we see that tuning *mtry* outperforms both tuning tree depth and tuning augmented bagging. We believe that this is due in part to the fact that the datasets we consider in our experiment span a wide range of SNRs. In the rightmost column of Table 3, we show the R^2 of the standard bagging ensembles (1000 full-depth trees) fit on each dataset, averaged across folds. In several cases, bagging ensembles achieve R^2 scores above 0.80, which suggests that the SNR of the dataset is high. Since *mtry* can reduce both bias

and variance, tuning this parameter appears to be effective across a variety of real-world datasets. In the rightmost column of Table 3, we report the best tuning method for each dataset in our experiment; across the majority of datasets, tuning *mtry* performs the best.

5.2.1 Default values for *mtry*

We conclude with some brief remarks on the default value for *mtry*. One main advantage of random forests—particularly when compared to other methods such as boosting—is that they are considered easier to use "off the shelf," requiring relatively little parameter tuning (see Efron and Hastie (2021), Chapter 17). This aligns with the classical view that random forests primarily reduce variance, whereas boosting ensembles mainly reduce bias (Efron and Hastie, 2021), and, as such, random forests are often fit with *mtry* set to the default value of 0.333 (Breiman, 2001; Efron and Hastie, 2021; Mentch and Zhou, 2020)

In our paper, we show that *mtry* controls both bias and variance reduction. As such, we would expect tuning *mtry* to improve the predictive performance of random forests on real world datasets, which span a range of SNRs, and we show here that this is the case. In the second from the right plot of Figure 23, we show the percent increase in R^2 between a random forest with the default $mtry = 0.333$ over a bagging ensemble. Compare this plot with the leftmost plot in Figure 23, which shows the percent increase in R^2 between a tuned random forest over bagging. We see that tuning *mtry* consistently improves the performance of the ensemble.

Recall from §5.1 that while reducing *mtry* can help random forests capture hidden patterns, doing so may also increase the influence of non-informative splits, which may increase bias. Now, consider non-informative splits in the context of individual decision trees in the ensemble, which have hierarchal structure. Influential non-informative splits in the top layers (for example depth < 3) of a decision tree may not significantly increase bias *if* subsequent splits in the deeper layer of the tree can correct for their effect. However, *mtry* typically applies to all splits in a tree regardless of depth, so this correction is unlikely to occur.

With this in mind, we propose a new default setting for *mtry* in random forests, called **variable mtry**, where we only randomize splits in the top layers of each tree in the ensemble. For each tree, we set $mtry = 0.333$ for splits of depth 1 and 2, and then we set $mtry = 1.0$ for the remainder of the deeper splits, until the tree is grown to full depth. We aim to detect hidden patterns in the top layers of a decision tree while letting lower layers correct for the non-informative splits introduced by setting $mtry < 1.0$.

We evaluate this default setting using the procedure discussed above. The rightmost plot in Figure 23 shows the percent increase in R^2 of our variable *mtry* method compared to standard bagging. We see that using the variable *mtry* method outperforms the default random forest method of setting $mtry = 0.333$ across all splits in the trees. In fact, our variable *mtry* default even outperforms tuning *mtry*. These results suggest that randomizing only the top layer splits in a random forest may allow the ensemble to capture hidden patterns while decreasing the impact of non-informative splits. Varying *mtry* by depth-layer presents an interesting direction for future research and may potentially yield new default parameter settings for random forests that improve predictive performance while preserving their ease of use.

6. Conclusion

In this paper, we show that random forests can reduce both bias and variance compared to bagging. Our findings help explain the observed success of random forests across a wide range of SNRs and allow us to re-examine the longstanding notion that random forests only reduce variance. We introduce the concept of hidden patterns, patterns in the data missed by bagging, and show that random forests can better capture these patterns by randomizing the features considered per split. Empirically, we show that this leads random forests reducing both bias and variance on datasets with hidden patterns. Using these findings, we analyze the impact of *mtry* on bias reduction in random forests and offer several practical insights towards parameter tuning in random forests.

7. Acknowledgments

We would like to thank the anonymous referees for their thoughtful comments that resulted in significant improvements in the manuscript. This research is funded in part by a grant from the Office of Naval Research (ONR-N00014-21-1-2841).

Appendix A.

Appendix A.1

SNR	Trimmed Max Nodes	Bagging/RF Max Nodes
0.05	175	200
0.085	160	200
0.145	155	200
0.247	150	200
0.42	145	200
0.715	140	200
1.216	90	200
2.071	80	200
3.525	55	200
6.0	45	200

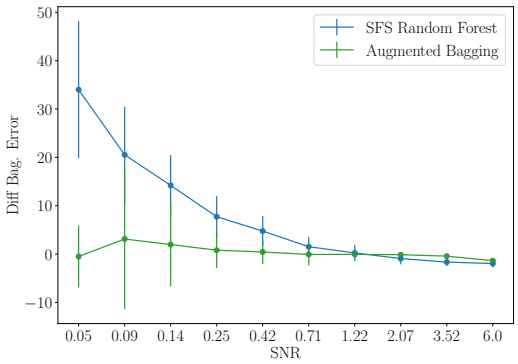
Table 4: Tuned *maxnodes* values, to match the effective DoF of a random forest, on the MARS dataset.

SNR	Trimmed Max Nodes	Bagging/RF Max Nodes
0.05	180	200
0.085	165	200
0.145	155	200
0.247	150	200
0.42	145	200
0.715	135	200
1.216	90	200
2.071	80	200
3.525	60	200
6.0	45	200

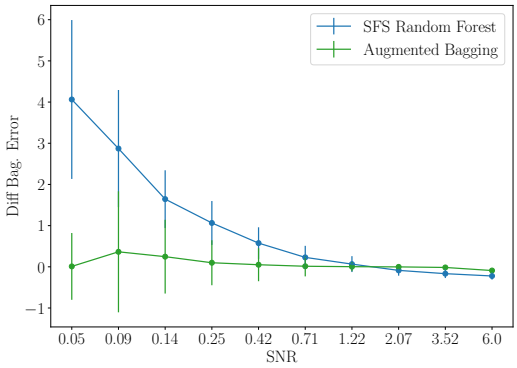
Table 5: Tuned *maxnodes* values, to match the effective DoF of a random forest, on the MARSadd dataset.

Appendix A.2

In this section, we repeat the experimental procedure discussed in §2.2.1, however, instead of our TRIM procedure, we assess the augmented bagging procedure discussed in Mentch and Zhou (2022). We tune q, the number of additional noise features to add, and r the correlation between the noise features and informative features, until the effective DoF of the augmented bagging ensemble matches that of a SFS random forest. We report the relative difference of each method compared to bagging in the plots below, for both the MARS and MARSadd example. From these plots, we observe that like for TRIM, augmented bagging reduces effective DoF without improving predictive performance on these examples.



(a) Relative Error of SFS random forests and augmented bagging ensembles on the MARS example.



(b) Relative Error of SFS random forests and augmented bagging ensembles on the MARSadd example.

Appendix B.

B.1 Proof of Proposition 2

We start from the LHS of the expression and plug in (8).

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{f}(X^{(i)}))^2 = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \frac{|\Psi_i|}{|T|}Y_i - \frac{1}{|T|}\sum_{t\in\Omega_i} L_t(X^{(i)})\right)^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left(Y_i(\frac{|T| - |\Psi_i|}{|T|}) - \frac{1}{|T|}\sum_{t\in\Omega_i} L_t(X^{(i)})\right)^2$$

We have that $|T| = |\Psi_i| + |\Omega_i|$.

$$= \frac{1}{n}\sum_{i=1}^{n}\left(\frac{|\Omega_i|}{|T|}Y_i - \frac{1}{|T|}\sum_{t\in\Omega_i} L_t(X^{(i)})\right)^2$$

$$= \frac{1}{n}\left(\frac{|\Omega_i|}{|T|}\right)^2\sum_{i=1}^{n}\left(Y_i - \frac{1}{|\Omega_i|}\sum_{t\in\Omega_i} L_t(X^{(i)})\right)^2$$

We plug in expression (9) to get the desired result. When the number of decision trees in a bagging ensemble/random forest is large we have:

$$\left(\frac{|\Omega_i|}{|T|}\right)^2 \approx 0.135,$$

which follows from the properties of the bootstrap.

Appendix C.

C.1 Empirical Study Models

- **friedman2_2d**

$$y = \sqrt{x_1^2 + \left(x_2x_3 - \frac{1}{x_2x_4 + 0.1}\right)^2} - 5\,\mathbb{1}(0.625 < x_5 \leq 0.65) + \epsilon$$

- **friedman3_2d**

$$y = \arctan\left(x_2x_3 - \frac{1}{x_2x_4}\right) + 2\,\mathbb{1}(0.625 < x_5 \leq 0.65) + \epsilon$$

- **polynomial_2d**

$$y = 2 + 3.5\,x_1 - 1.2\,x_1^2 + 4\,x_2 + 0.8\,x_2^3 - 5\,\mathbb{1}(0.625 < x_6 \leq 0.65) - 4\,\mathbb{1}(0.575 < x_7 \leq 0.60) - 6\,\mathbb{1}(0.58 < x_8 \leq 0.60) + \epsilon$$

- **polynomial_box**

$$y = 2 + 3.5\,x_1 - 1.2\,x_1^2 + 4\,x_2 + 0.8\,x_2^3 - 10\,\mathbb{1}(0.625 < x_6 \le 0.70,\ 0.625 < x_7 \le 0.70) + \epsilon$$

- **polynomial_spike**

$$\mathrm{G1}(x_5) = 6\,\exp\!\Big(-\tfrac{(x_5-0.28)^2}{2\,(0.005)^2}\Big),$$
$$\mathrm{G2}(x_5) = 6\,\exp\!\Big(-\tfrac{(x_5-0.30)^2}{2\,(0.005)^2}\Big),$$
$$y = 2 + 3.5\,x_1 - 1.2\,x_1^2 + 4\,x_2 + 0.8\,x_2^3 - \mathrm{G1}(x_5) + \mathrm{G2}(x_5) + \epsilon$$

- **linear_2d**

$$y = 0.3\,x_1 - 0.5\,x_2 + 0.1\,x_3 + 0.1\,x_4 + x_5 - \mathbb{1}(0.625 < x_6 \le 0.65) - 2\,\mathbb{1}(0.575 < x_7 \le 0.60) - 0.5\,\mathbb{1}(0.58 < x_8 \le 0.60) + \epsilon$$

- **linear_box**

$$y = 0.3\,x_1 - 0.5\,x_2 + 0.1\,x_3 + 0.1\,x_4 + x_5 - 3\,\mathbb{1}(0.63 < x_6 \le 0.65,\ 0.63 < x_7 \le 0.65) + \epsilon$$

- **linear_spike**

$$\mathrm{G1}(x_5) = 4\,\exp\!\Big(-\tfrac{(x_5-0.28)^2}{2\,(0.005)^2}\Big),$$
$$\mathrm{G2}(x_5) = 4\,\exp\!\Big(-\tfrac{(x_5-0.30)^2}{2\,(0.005)^2}\Big),$$
$$y = 0.3\,x_1 - 0.5\,x_2 + 0.1\,x_3 + 0.1\,x_4 + x_5 - \mathrm{G1}(x_5) + \mathrm{G2}(x_5) + \epsilon$$

- **hmars_2d**

$$y = 10\,\sin(\pi\,x_1x_2) + 20\,(x_3-0.05)^2 + 10\,x_4 + 5\,x_5 - 30\,\mathbb{1}(0.60 < x_6 \le 0.65) - 35\,\mathbb{1}(0.55 < x_7 \le 0.60) + \epsilon$$

- **hmars_box**

$$y = 10\,\sin(\pi\,x_1x_2) + 20\,(x_3-0.05)^2 + 10\,x_4 + 5\,x_5 - 250\,\mathbb{1}(0.65 < x_6 \le 0.70,\ 0.65 < x_7 \le 0.70) + \epsilon$$

- **hmars_spike**

$$\mathrm{G1}(x_6) = 100\,\exp\!\Big(-\tfrac{(x_6-0.28)^2}{2\,(0.005)^2}\Big),$$
$$\mathrm{G2}(x_6) = 100\,\exp\!\Big(-\tfrac{(x_6-0.30)^2}{2\,(0.005)^2}\Big),$$
$$y = 10\,\sin(\pi\,x_1x_2) + 20\,(x_3-0.05)^2 + 10\,x_4 + 5\,x_5 - \mathrm{G1}(x_6) + \mathrm{G2}(x_6) + \epsilon$$

- **hmars_add_box**

$$y \;=\; 0.1\,\exp(4\,x_1) \;+\; \frac{4}{1+\exp[-20\,(x_2-0.5)]} \;+\; 3\,x_3 \;+\; 2\,x_4 \;+\; x_5 \\ -\; 105\,\mathbb{1}(\,0.45 < x_6 \leq 0.55,\; 0.45 < x_7 \leq 0.55) \;+\; \epsilon$$

- **hmars_add_2d**

$$y \;=\; 0.1\,\exp(4\,x_1) \;+\; \frac{4}{1+\exp[-20\,(x_2-0.5)]} \;+\; 3\,x_3 \;+\; 2\,x_4 \;+\; x_5 \\ -\; 10\,\mathbb{1}(\,0.60 < x_6 \leq 0.65) \;-\; 7.5\,\mathbb{1}(\,0.55 < x_7 \leq 0.60) \;+\; \epsilon$$

- **hmars_add_spike**

$$\mathrm{G1}(x_6) = 10\,\exp\Big(-\tfrac{(x_6-0.28)^2}{2\,(0.005)^2}\Big), \\ \mathrm{G2}(x_6) = 10\,\exp\Big(-\tfrac{(x_6-0.30)^2}{2\,(0.005)^2}\Big),$$

$$y \;=\; 0.1\,\exp(4\,x_1) \;+\; \frac{4}{1+\exp[-20\,(x_2-0.5)]} \;+\; 3\,x_3 \;+\; 2\,x_4 \;+\; x_5 \\ -\; \mathrm{G1}(x_6) \;+\; \mathrm{G2}(x_6) \;+\; \epsilon$$

Appendix D.

D.1

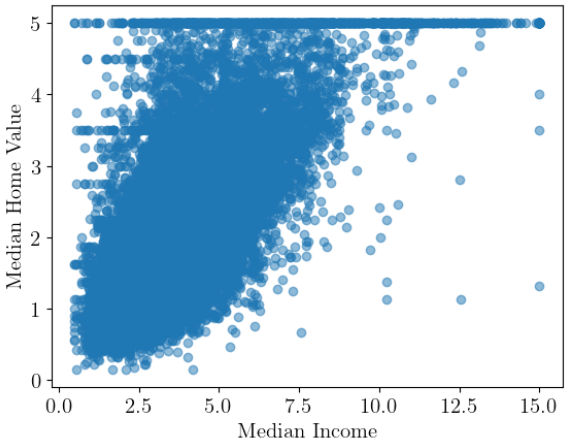


Figure 25: Median home value vs. median income for the California housing dataset.

D.2

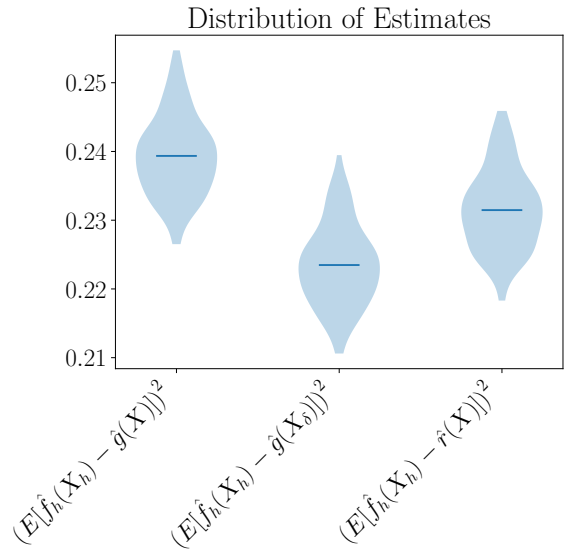


Figure 26: Distribution of estimated quantities in the California housing example.

Appendix E.

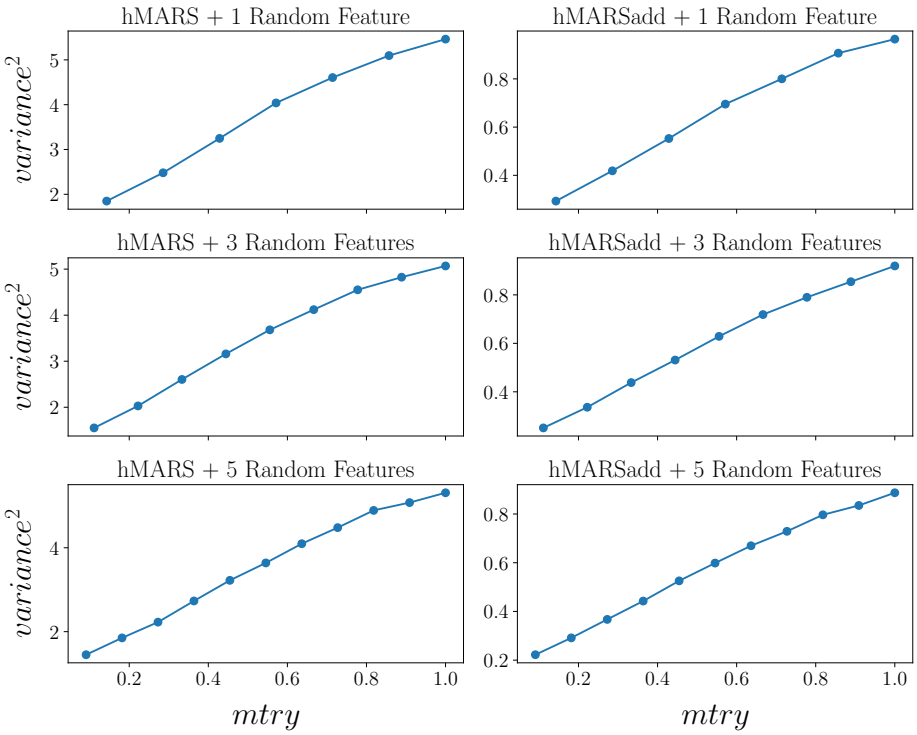


Figure 27: Variance component of bias-variance decomposition for experiment in §5.

Appendix F.

Dataset Name	R^2 **Bagging**	**Best Tuned** *mtry*
autoMpg	0.818	0.100
no2	0.602	0.787
boston	0.877	0.542
stock	0.985	0.493
socmob	0.804	0.362
spacega	0.662	0.902
abalone	0.546	0.738
winequality	0.528	0.493
wind	0.783	0.869
puma32H	0.933	0.885
bank32nh	0.513	0.575
cpusmall	0.977	0.493
kin8nm	0.718	0.656
pol	0.988	0.673
elevators	0.839	0.836
houses	0.821	0.656
house16H	0.635	0.624
mv	0.982	0.885

Table 6: Tuned values of *mtry*, averaged across folds, for each dataset used in the experiments in §5.2.

References

Boubekeur Baba and Güven Sevil. Predicting ipo initial returns using random forest. *Borsa Istanbul Review*, 20(1):13–23, 2020.

Ash Booth, Enrico Gerding, and Frank McGroarty. Automated trading with performance weighted random forests and seasonality. *Expert Systems with Applications*, 41(8):3651–3661, 2014.

Leo Breiman. Bagging predictors. *Machine learning*, 24:123–140, 1996.

Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

Leo Breiman. *Classification and regression trees*. Routledge, 2017.

Peter Bühlmann and Bin Yu. Analyzing bagging. *The annals of Statistics*, 30(4):927–961, 2002.

Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168, 2006.

Mariela Cerrada, Grover Zurita, Diego Cabrera, René-Vinicio Sánchez, Mariano Artés, and Chuan Li. Fault diagnosis in spur gears based on genetic algorithm and random forest. *Mechanical Systems and Signal Processing*, 70:87–103, 2016.

Ramón Díaz-Uriarte and Sara Alvarez de Andrés. Gene selection and classification of microarray data using random forest. *BMC bioinformatics*, 7:1–13, 2006.

Pedro Domingos. A unified bias-variance decomposition. In *Proceedings of 17th international conference on machine learning*, pages 231–238. Morgan Kaufmann Stanford, 2000.

Pedro M Domingos. Why does bagging work? a bayesian account and its implications. In *KDD*, pages 155–158. Citeseer, 1997.

Bradley Efron and Trevor Hastie. *Computer age statistical inference, student edition: algorithms, evidence, and data science*, volume 6. Cambridge University Press, 2021.

Bradley Efron and Robert Tibshirani. Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical science*, pages 54–75, 1986.

Jerome H Friedman. Multivariate adaptive regression splines. *The annals of statistics*, 19(1): 1–67, 1991.

Jerome H Friedman. On bias, variance, 0/1—loss, and the curse-of-dimensionality. *Data mining and knowledge discovery*, 1:55–77, 1997.

Yves Grandvalet. Bagging equalizes influence. *Machine Learning*, 55:251–270, 2004.

Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

Andy Liaw and Matthew Wiener. *randomForest: Breiman and Cutler's Random Forests for Classification and Regression*, 2002. URL `https://cran.r-project.org/package=randomForest`. R package version 4.7-1.

Yi Lin and Yongho Jeon. Random forests and adaptive nearest neighbors. *Journal of the American Statistical Association*, 101(474):578–590, 2006.

Chengwei Liu, Yixiang Chan, Syed Hasnain Alam Kazmi, and Hao Fu. Financial fraud detection model: Based on random forest. *International journal of economics and finance*, 7(7), 2015.

Christoph Lohrmann and Pasi Luukka. Classification of intraday s&p500 returns with a random forest. *International Journal of Forecasting*, 35(1):390–407, 2019.

Rahul Mazumder. Discussion of "best subset, forward stepwise or lasso? analysis and recommendations based on extensive comparisons". *Statistical Science*, 35(4), 2020.

Lucas Mentch and Siyu Zhou. Randomization as regularization: A degrees of freedom explanation for random forest success. *The Journal of Machine Learning Research*, 21(1): 6918–6953, 2020.

Lucas Mentch and Siyu Zhou. Getting better from worse: Augmented bagging and a cautionary tale of variable importance. *Journal of Machine Learning Research*, 23(224): 1–32, 2022.

R Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297, 1997.

Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. URL `https://jmlr.org/papers/v12/pedregosa11a.html`.

Fatemeh Saki, Abhishek Sehgal, Issa Panahi, and Nasser Kehtarnavaz. Smartphone-based real-time classification of noise signals using subband features and random forest classifier. In *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 2204–2208. IEEE, 2016.

Abraham J Wyner, Matthew Olson, Justin Bleich, and David Mease. Explaining the success of adaboost and random forests as interpolating classifiers. *The Journal of Machine Learning Research*, 18(1):1558–1590, 2017.

Lei Xu, Yunfu Wang, Lin Mo, Yongfan Tang, Feng Wang, and Changjun Li. The research progress and prospect of data mining methods on corrosion prediction of oil and gas pipelines. *Engineering Failure Analysis*, 144:106951, 2023.

Siyu Zhou and Lucas Mentch. Trees, forests, chickens, and eggs: when and why to prune trees in a random forest. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 16(1):45–64, 2023.