How do Random Forests work?

Authored by

stats writer

December 18, 2025

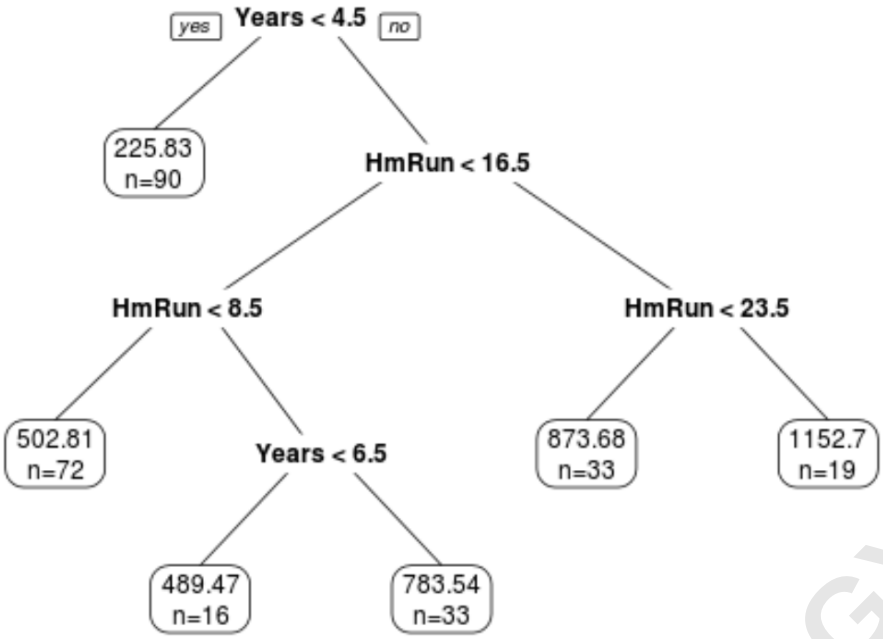
The Random Forest algorithm stands as one of the most powerful and widely used supervised learning methods in modern data science. It functions by constructing a multitude of individual decision trees from randomly selected subsets of the training data. The core principle involves each tree making a prediction, followed by the aggregation--typically averaging (for regression) or voting (for classification)--of these results to yield a final, highly reliable output. This ensemble method is particularly effective because it significantly reduces the model's variance, thereby mitigating the risk of **overfitting** while simultaneously improving predictive accuracy across diverse datasets.

Understanding how Random Forests achieve this robust performance requires first examining the foundational concepts: single decision trees and the process known as **Bagging**. When dealing with real-world phenomena where the relationship between input features (predictors) and the target outcome (response variable) is complex and non-linear, simple linear models often fall short. Decision trees provide an intuitive, non-linear pathway, but they possess inherent statistical limitations that Random Forests are specifically designed to overcome.

The Foundation: Classification and Regression Trees (CART)

When modeling complex relationships between a set of features (predictor variables) and a target variable (response variable), we frequently turn to non-linear modeling techniques. A prominent example of this is the use of **Classification and Regression Trees** (often abbreviated CART). These models recursively partition the feature space into distinct, non-overlapping regions, ultimately creating a simple, hierarchical structure--the decision tree--that predicts the value of a response variable based on the input features.

The primary appeal of single decision trees lies in their exceptional interpretability and visual clarity, making the logic behind the prediction transparent. However, this simplicity comes at a cost: they tend to suffer from **high variance**. High variance implies that small changes in the training data can lead to dramatically different tree structures and, consequently, highly variable predictions. For instance, if a dataset were split into two halves and a decision tree were trained on both, the resulting models could produce substantially different outputs, indicating poor generalization capabilities.



Example of a regression tree that uses years of experience and average home runs to predict the salary of a professional baseball player.

Reducing Variance Through Bagging (Bootstrap Aggregation)

To address the high variance issue inherent in single decision trees, statisticians developed an ensemble technique called **Bagging**, or Bootstrap Aggregation. This method aims to stabilize the predictive model by introducing randomness in the data sampling process. By averaging the results of many trees trained on slightly different versions of the training data, Bagging significantly reduces the overall model variance, leading to a much better test error rate compared to any single, unstable decision tree.

The operational process of Bagging involves the following steps:

Take b bootstrapped samples (samples drawn with replacement) from the original dataset.
Build an independent decision tree for each bootstrapped sample.
Average the predictions of each tree to come up with a final model.

The benefit of this approach is clear: a bagged model typically offers a strong improvement in test error rate compared to a single, high-variance decision tree. However, this method introduces a new challenge related to prediction correlation.

A significant limitation of standard Bagging arises if the dataset happens to contain a very strong predictor variable. In this case, most or all of the bootstrapped trees will inevitably use this

dominant predictor for the first split. This results in trees that are structurally similar to each other and, crucially, have highly correlated predictions. Consequently, when we average the predictions of this collection of correlated trees, the resulting model does not sufficiently reduce the variance compared to a single tree, limiting the performance improvement.

How Random Forests Achieve Decorrelation

One powerful way to circumvent the correlation issue found in standard Bagging models is to use the method known as **Random Forests**. Similar to Bagging, Random Forests also begin by taking b bootstrapped samples from the original dataset. However, the procedure introduces a critical modification during the tree construction phase that ensures the resulting collection of trees is decorrelated.

The key innovation is that when building a decision tree for each bootstrapped sample, every time a split in a tree is considered, only a random sample of m predictors is considered as split candidates from the full set of p predictors. By randomly restricting the feature set available at each split, the influence of dominant predictor variables is diluted, forcing the trees to rely on different sets of features. This guarantees a higher degree of independence between the individual trees in the forest.

Because the collection of trees in a Random Forest is **decorrelated**, when we take the average predictions of each tree to come up with a final model, the overall variability is significantly less than that of a bagged model. This reduction in variance results in a consistently lower test error rate. In practice, a common guideline when using Random Forests is to set the number of split candidates m to the square root of the total number of predictors p (i.e., $m = sqrt{p}$). For example, if a dataset contains $p = 25$ total features, we would typically consider $m = sqrt{25} = 5$ predictors as potential split candidates at each node.

The Full Random Forest Algorithm

The complete method that Random Forests use to build a robust and decorrelated model is summarized in the following steps:

Take b bootstrapped samples from the original training dataset.

For each bootstrapped sample, grow an unpruned decision tree using the following modified approach:

When building the tree, each time a split is considered, only a random sample of m predictors is considered as split candidates from the full set of p predictors.

Average the predictions of all b individual trees (or take a majority vote for classification) to form

the final model.

By using this method, the resulting model balances the reduction in variance provided by ensemble learning with the necessity of ensuring diversity among the ensemble members, leading to optimal predictive performance.

Technical Note: The Equivalence to Bagging

It is interesting to note the boundary condition: if we choose $m = p$ (meaning all predictors are considered as split candidates at each node), the feature randomization step is bypassed entirely. When m equals p, the Random Forest algorithm is computationally equivalent to simply using **Bagging**.

Out-of-Bag Error Estimation

A major benefit of ensemble methods that rely on bootstrapping is the ability to estimate the model's generalization error internally using **out-of-bag estimation**. This eliminates the need for external validation sets or costly cross-validation procedures.

When a bootstrapped sample is drawn, statistical theory dictates that approximately one-third of the observations from the original dataset are not included in that particular sample. These excluded data points are known as **out-of-bag (OOB) observations**. Since these OOB observations were not used to train the corresponding decision tree, they serve as a unique, unbiased test set for that specific tree.

To calculate the overall test error, we predict the value for the i-th observation in the original dataset by averaging the predictions from all trees in which that specific observation was OOB. This procedure is applied to all n observations in the original dataset, allowing us to compute an overall error rate. This error rate is considered a valid and reliable estimate of the test error. The core benefit of using OOB estimation is its speed; it is much quicker than standard validation methods like k-fold cross-validation, especially when working with massive datasets where computational efficiency is paramount.

The Pros & Cons of Random Forests

Random Forests are highly regarded for their predictive power, but practitioners must weigh their advantages against their limitations before deployment.

Random Forests offer the following significant **benefits**:

Accuracy Superiority: In most applications, Random Forests deliver superior accuracy compared to standard bagged models and single decision trees due to their effective variance reduction.

Robustness: The algorithm is naturally robust to noise and outliers in the data, as the ensemble averaging dampens the effect of anomalies.

Ease of Use: No extensive pre-processing, such as feature scaling or standardization, is required to implement Random Forests, simplifying the data preparation phase.

However, Random Forests also come with the following potential **drawbacks:**

Low Interpretability: By aggregating hundreds of complex trees, the model loses the transparency of a single decision tree, making it difficult to trace the exact relationship between features and predictions.

Computational Intensity: Training and maintaining a large forest can be computationally demanding and slow, particularly when dealing with datasets that have millions of observations or thousands of features.

In contemporary data science, Random Forests are predominantly utilized when maximizing predictive accuracy is the primary objective, often making their lack of easy interpretability an acceptable trade-off for their robust performance.