**Random Forests Algorithm explained with a real-life example and some Python code**

**Random Forests is a Machine Learning algorithm that tackles one of the biggest problems with** [**Decision Trees**](https://towardsdatascience.com/decision-tree-classifier-explained-in-real-life-picking-a-vacation-destination-6226b2b60575)**: variance.**

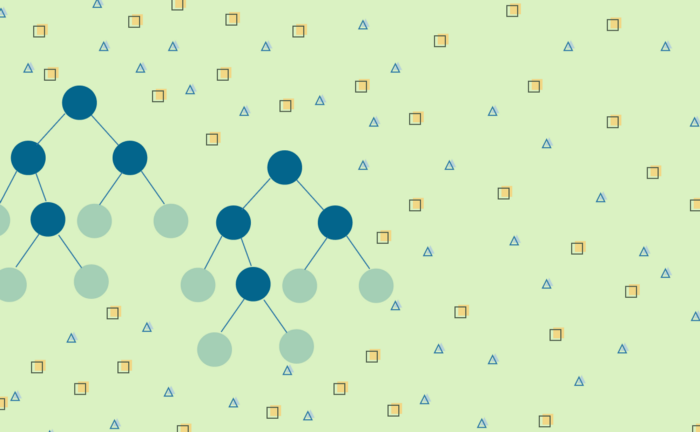


Image by author.

*This is article number two in a series dedicated to Tree Based Algorithms, a group of widely used Supervised Machine Learning Algorithms.*

*The first article was about* [*Decision Trees*](https://towardsdatascience.com/decision-tree-classifier-explained-in-real-life-picking-a-vacation-destination-6226b2b60575)*. The next, and last article in this series, explores Gradient Boosted Decision Trees. Everything explained with real-life examples and some Python code.*

*Stay tuned!*

Random Forests is a Machine Learning algorithm that tackles one of the biggest problems with [Decision Trees](https://towardsdatascience.com/decision-tree-classifier-explained-in-real-life-picking-a-vacation-destination-6226b2b60575): **variance**.

Even though Decision Trees is simple and flexible, it is [greedy algorithm](https://en.wikipedia.org/wiki/Greedy_algorithm). It focuses on optimizing for the node split at hand, rather than taking into account how that split impacts the entire tree. A greedy approach makes Decision Trees run faster, but makes it prone **overfitting**.

An overfit tree is highly optimized to predicting the values in the training dataset, resulting in a learning model with **high-variance**.

How you calculate variance in a Decision Tree depends on the problem you’re solving.

In a Regression task you can calculate actual [variance](https://en.wikipedia.org/wiki/Variance) of the prediction compared to the true targets. If the tree produces results that are too far off from its true targets, it has high-variance and therefore, it is overfit.

A highly overfit tree has high-variance. That means its predictions are extremely far off from the actual targets.

But in a Classification task, you can’t use the same approach. The way to detect if the Decision Tree is overfit is by looking at the test error. If the tree has a high test error, meaning it’s not good at classifying observations it wasn’t trained on, it is overfit.

In Regression tasks overfitting is detected by high-variance, while in Classification tasks it’s by a high generalization error.

To address overfitting, and reduce the variance in Decision Trees, [Leo Breiman](https://en.wikipedia.org/wiki/Leo_Breiman) developed the [Random Forests](https://en.wikipedia.org/wiki/Random_forest) algorithm[1]. This was an innovative algorithm because it utilized, for the first time, the statistical technique of [Bootstrapping](https://en.wikipedia.org/wiki/Bootstrapping_(statistics)) and combined the results of training multiple models into a single, more powerful learning model.

But before you see Random Forests in action, and code, let’s take a detour to explore what makes Random Forests unique.

**Bagging: Bootstrap Aggregation**

**Bagging**, short for [Bootstrap Aggregation](https://en.wikipedia.org/wiki/Bootstrap_aggregating), is a technique developed by Leo Breiman, with the goal of reducing the variance of a learning model. Bagging is also model agnostic, so regardless of type of model you’re using, the process is the same.

The **Bootstrapping** part of Bagging refers to the resampling method in which several random samples are drawn with replacement from a dataset[3]. As a consequence, Bootstrapping creates multiple, smaller random datasets drawn from the same distribution.

Each of the bootstrapped datasets is used to train a model, and the outputs are then **Aggregated** into one final result.

But aggregation also means different things, depending on the type of problem you’re solving.

When you’re working on a Regression problem, aggregation means *averaging* the results of each observation, across all models. While in Classification aggregation means picking the most common class for each observation, like doing a *majority vote*.

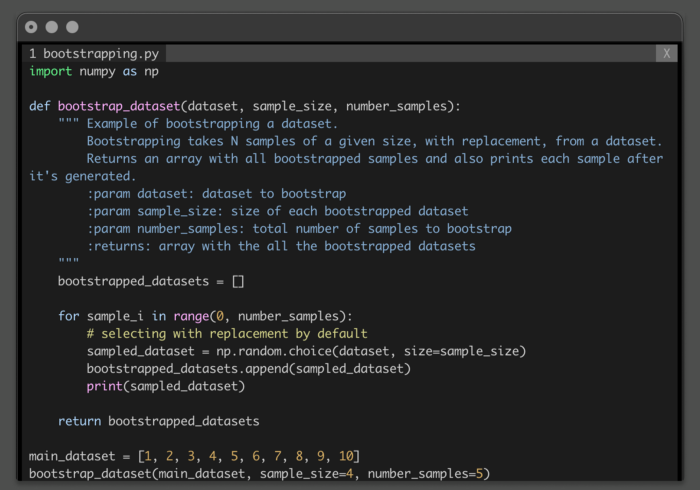
This is impressivebut, *how does this actually help reduce model variance?*

Each model is trained on a *different* dataset, because they’re bootstrapped. So inevitably, each model will make *different* mistakes, and have a distinct error and variance. Both the error and variance get reduced in the aggregation step where, literally in the case of Regression, they are *averaged* out.

And because you end up with a single model, which combines the output of multiple models, Bagging is called and **ensemble technique**.

**Bootstrapping**

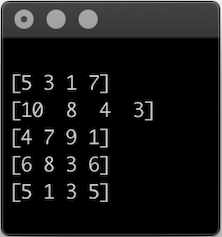
With Bootstrapping you’re following the motto of *doing more with less*. You’re taking one dataset and multiplying it into several smaller random datasets.



Code sample that bootstraps a dataset.

The initial dataset only had 10 elements, but you ended producing 5 sampled datasets of size 4.

In a way, **you multiplied the dataset**, going from just 10 observations to 4x5=20 observations, when you sum up the individual bootstrapped datasets.



*Result of bootstrapping a dataset, generating 5 samples with 4 elements each.*

The reason why Bootstrapping works is because you’re **sampling** **with** **replacement**.

You can *pick* the same datapoint multiple times, like in the last two samples, but each sampled dataset is slightly different from the previous one.

Without replacement you’d quickly run out of data points, and would only be able to generate a limited number of samples.

This detour was focused on what makes Random Forest unique.

Let’s get back to the main topic, *how Random Forests reduces model variance.*

**Random Forests**

Random Forests was developed specifically to address the problem of high-variance in Decision Trees. Like the name suggests, you’re not training a single Decision Tree, you’re training an entire forest! In this case, a forest of Bagged Decision Trees.

At a high-level, in *pseudo-code*, Random Forests algorithm follows these steps:

1. Take the original dataset and create *N* bagged samples of size *n*, with *n* smaller than the original dataset.
2. Train a Decision Tree with each of the *N* bagged datasets as input. But, when doing a node split, don’t explore all features in the dataset. Randomly select a smaller number, *M* features, from all the features in training set. Then pick the best split using impurity measures, like [Gini Impurity](https://en.wikipedia.org/wiki/Decision_tree_learning#Gini_impurity) or Entropy.
3. Aggregate the results of the individual decision trees into a single output.
4. Average the values for each observation, produced by each tree, if you’re working on a Regression task.
5. Do a majority vote across all trees, for each observation, if you’re working on a Classification task.

While *Forest* part of Random Forests refers to training multiple trees, the *Random* part is present at two different points in the algorithm.

There’s the randomness involved in the Bagging process. But then, you also pick a random subset of features to evaluate the node split. This is what guarantees that each tree is different and, therefore, ensures each model produces a slightly different result.

And while you could be thinking that randomly sampling features at each split introduces yet another [hyperparameter](https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning)) you might need to tune, that’s not the case. The model takes care of it for you!

You can certainly tune this hyperparameter. However, there’s mathematical consensus about randomly picking a number of features that is equal to the square root of the total number of available features in the dataset[2].

**Why use Random Forests?**

Random Forests has several advantages, when compared to training a single Decision Tree.

**All the advantages of Decision Trees, but more powerful**

At the core of this algorithm is a Decision Tree so, Random Forests shares all its advantages.

It’s a data robust algorithm, being able to handle different types of data, and doesn’t require any data preprocessing.

The true potential of Random Forests comes from combining the results different Decision Trees.

**Each model is different**

Like Decision Trees, the algorithm optimizes for the local split. But, instead of exploring all possible splits for each feature in the dataset, it randomly picks a subset of those features.

This reduces the number of outcomes the algorithm needs to evaluate at each split and, it makes each trained tree is slightly different.

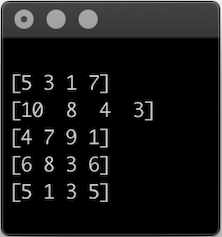
**No holdout set required**

In Machine Learning you typically split the dataset into training and testing sets, in an effort to evaluate model performance with observations it has never seen before.

This becomes a challenging problem when you have a small dataset or the cost, and effort, of collecting more data is high.

But with Random Forests you can use the entire dataset to train and evaluate the model. The Bagging process takes care of it for you! Since you’re generating N smaller, random datasets picked with replacement, there’s always going to be a set of points that was not used to create the tree.

Take a look at the earlier example, which bootstrapped the dataset [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].



*Result of bootstrapping a dataset, generating 5 samples with 4 elements each.*

If you were to use each of these samples to train a Decision Tree, there are observations the trees simply *don’t know about,* because they were not part of their training set.

With Random Forests you can use the entire dataset to train the model and calculate the test error directly from its results. You don’t need to set aside part of the dataset just for testing.

This new type of error, calculated directly from its results of the trained trees, is called the [Out-of-bag Error](https://en.wikipedia.org/wiki/Out-of-bag_error).

**🛩🏝 Reducing model variance with Random Forests**

Planning a vacation is challenging. This time you’re going to put your Machine Learning skills into practice and get an algorithmic opinion on what should be your next vacation destination.

Whenever you start planning a vacation, you always take into account:

* Duration of the vacation,
* Personal budget,
* Weather forecast,
* If your extended family is joining,
* If you’re feeling adventurous and want to explore new places.

Decision Trees are said to mimic how humans make decisions. But as you’ve seen on the [previous article of this series](https://towardsdatascience.com/decision-tree-classifier-explained-in-real-life-picking-a-vacation-destination-6226b2b60575), a single Decision Tree is not a very powerful predictor.

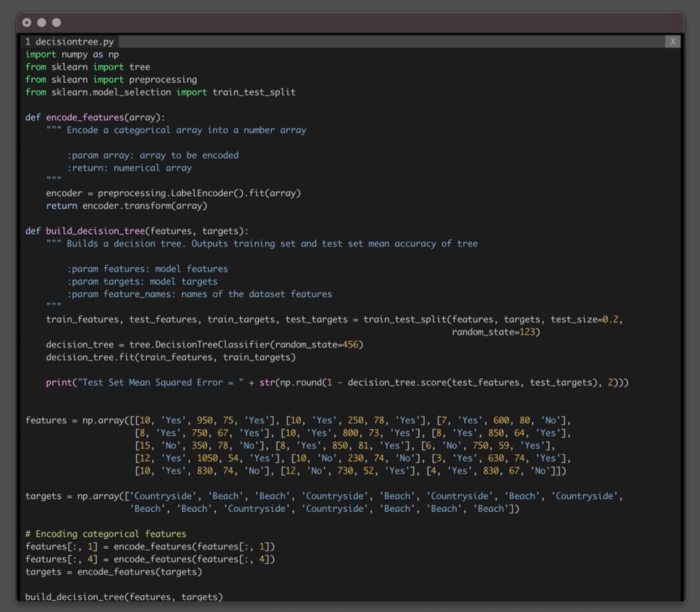
Random Forests is an improvement on Decision Trees, but is it really that much better?

There’s only one way to find out!

But first, take another look at the performance of a single Decision Tree.

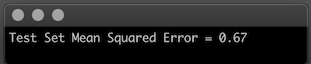
**Test error for a single Decision Tree**

If you train just one Decision Tree, the test error is approximately 70%.



Code sample to train a single DecisionTree to predict your next vacation destination.

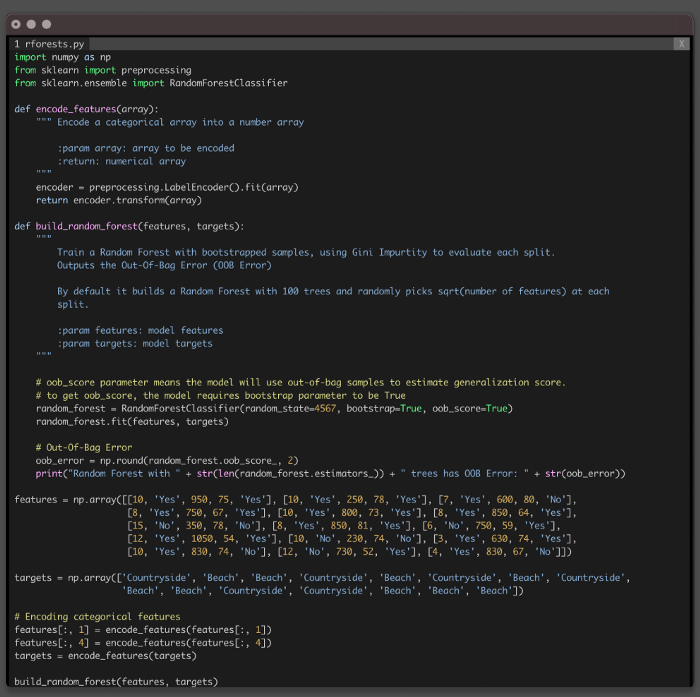
The test error, also known as generalization error, is considerably high. A sign of high variance.



Test Error from the Decision Tree model.

Ok, you’re convinced! Decision Trees is definitely not the best algorithm to help you in the critical task of picking the next vacation destination.

Let’s train a Random Forests model using [ScikitLearn](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html), completely *out-of-the-box* and without any hyperparameter tuning.



Code to train a Random Forest to train to predict your next vacation destination.

Right away Random Forests reduced the Test Error by 30%!

This is an impressive result, comparing to the performance of a single Decision Tree.



Out-of-Bag Error, i.e., test error, from the Random Forests model.

One thing you noticed was that, by default the algorithm trains and combines the result of 100 bagged trees. This is one of the many hyperparameters you can tune in [Random Forests](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html).

Curiosity just kicked-in!

*Why 100 trees?*

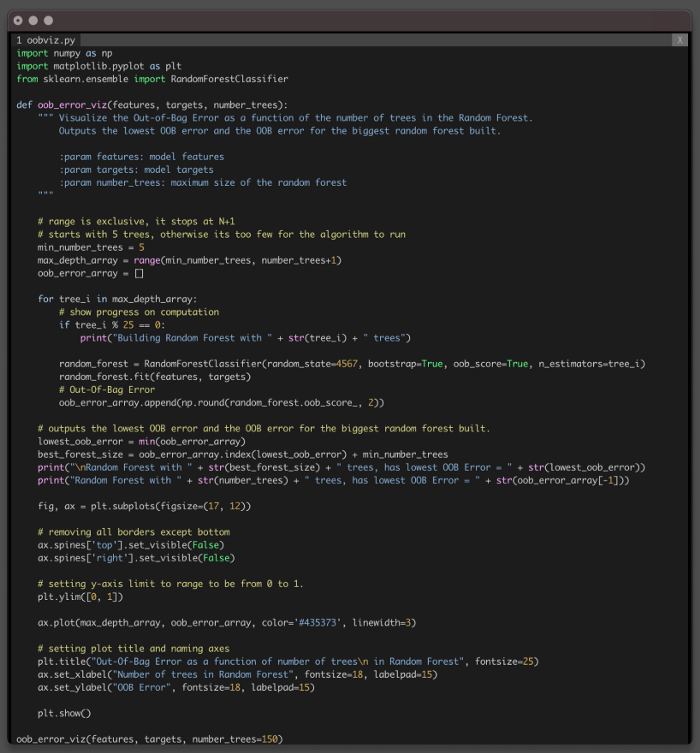
*Would the algorithm perform better or worse with a different number of trees?*

*How much can you reduce the generalization error?*

**Find the Optimal Number of Trees**

Instead of tuning this hyperparameter by trail and error, you decided plot the *Out-Of-Bag-Error* as a function of the number of trees in the Random Forest.

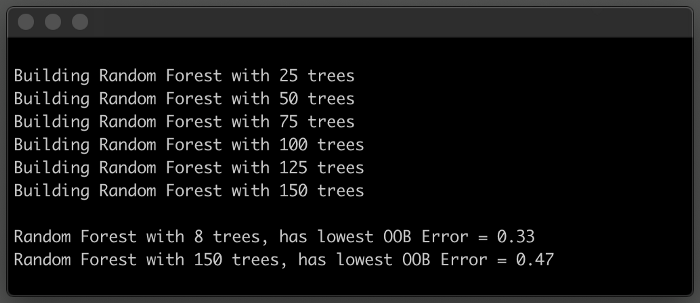
You settled on training the arbitrary number of 150 trees. But you’re starting with only 5 and slowly making your way up to 150, building a new model, each time with one additional tree.



Code sample to plot the Out-Of-Bag error as the number of trees in the Random Forest.

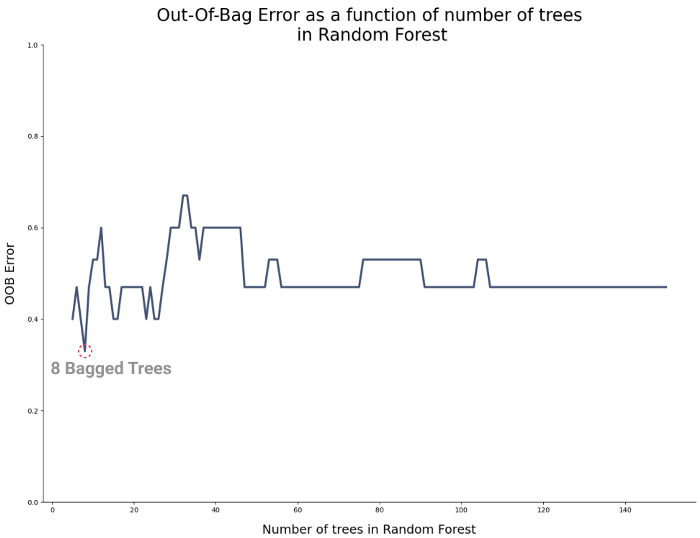
Training a large forest doesn’t improve the model.

In the process you’ve found the performance *sweet spot* for your task. You only need 8 trees in the Random Forests to drop the generalization error to 33%.



Output of plotting the *Out-of-Bag-Error* as a function of the number of trees in the Random Forest.

Looking at the plot, it’s easier to see the Out-Of-Bag Error getting worse past the *sweet-spot* of 8 trees. Then the model hits a *performance plateau*, when the *forest* gets bigger than 50 trees.



**Conclusion**

You successfully reduced the variance of your model!

Now you can be more confident about using Random Forests to predict your next vacation destination. Instead of training a single Decision Tree.

This was a simple example, but you could see the generalization error decline significantly, from 67% to 33%.

Random Forests is a powerful algorithm, but it has a clear disadvantage. You can’t *see* how the model makes decisions.

A cool aspect of Decision Trees is that, after the model is trained, you can actually visualize how the model makes decisions. With a small enough tree you can trace the entire *decision-making* process for a new observation.

But with Random Forests, you trade-off interpretability for performance.

As soon as you start combining the decisions of multiple trees, the *decision process* gets much more complex and impossible to visualize.