The Random Forest Algorithm

Niklas Donges Follow
Feb 22, 2018 · 8 min read

Random Forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because it's simplicity and the fact that it can be used for both classification and regression tasks. In this post, you are going to learn, how the random forest algorithm works and several other important things about it.

Table of Contents:

- Feature Importance
- Difference between Decision Trees and Random Forests
- Important Hyperparameters (predictive power, speed)
- Advantages and Disadvantages
- Use Cases
- Summary

How it works:

Random Forest is a supervised learning algorithm. Like you can already see from it’s name, it creates a forest and makes it somehow random. The „forest“ it builds, is an ensemble of Decision Trees, most of the time trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

To say it in simple words: Random forest builds multiple decision trees and merges them together to get a more *accurate and stable prediction.*

One big advantage of random forest is, that it can be used for both classification and regression problems, which form the majority of current machine learning systems. I will talk about random forest in classification, since classification is sometimes considered the building block of machine learning. Below you can see how a random forest would look like with two trees:

Random Forest has nearly the same hyperparameters as a decision tree or a bagging classifier. Fortunately, you don’t have to combine a decision tree with a bagging classifier and can just easily use the classifier-class of Random Forest. Like I already said, with Random Forest, you can also deal with Regression tasks by using the Random Forest regressor.

Random Forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

Therefore, in Random Forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. You can even make trees more random, by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

Real Life Analogy:

Imagine a guy named Andrew, that want's to decide, to which places he should travel during a one-year vacation trip. He asks people who know him for advice. First, he goes to a friend, tha asks Andrew where he traveled to in the past and if he liked it or not. Based on the answers, he will give Andrew some advice.

This is a typical decision tree algorithm approach. Andrews friend created rules to guide his decision about what he should recommend, by using the answers of Andrew.

Afterwards, Andrew starts asking more and more of his friends to advise him and they again ask him different questions, where they can derive some recommendations from. Then he chooses the places that where recommend the most to him, which is the typical Random Forest algorithm approach.

Feature Importance:

Another great quality of the random forest algorithm is that it is very easy to measure the relative importance of each feature on the prediction. Sklearn provides a great tool for this, that measures a features importance by looking at how much the tree nodes, which use that feature, reduce impurity across all trees in the forest. It computes this score automatically for each feature after training and scales the results, so that the sum of all importance is equal to 1.

If you don’t know how a decision tree works and if you don’t know what a leaf or node is, here is a good description from Wikipedia: In a decision tree each internal node represents a “test” on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). A node that has no children is a leaf.

Through looking at the feature importance, you can decide which features you may want to drop, because they don’t contribute enough or nothing to the prediction process. This is important, because a general rule in machine learning is that the more features you have, the more likely your model will suffer from overfitting and vice versa.

Below you can see a table and a visualization that show the importance of 13 features, which I used during a supervised classification project with the famous Titanic dataset on kaggle. You can find the whole project here.

Difference between Decision Trees and Random Forests:

Like I already mentioned, Random Forest is a collection of Decision Trees, but there are some differences.

If you input a training dataset with features and labels into a decision tree, it will formulate some set of rules, which will be used to make the predictions.

For example, if you want to predict whether a person will click on an online advertisement, you could collect the ad's the person clicked in the past and some features that describe his decision. If you put the features and labels into a decision tree, it will generate some rules. Then you can predict whether the advertisement will be clicked or not. In comparison, the Random Forest algorithm randomly selects observations and features to build several decision trees and then averages the results.

Another difference is that „deep“ decision trees might suffer from overfitting. Random Forest prevents overfitting most of the time, by creating random subsets of the features and building smaller trees using these subsets. Afterwards, it combines the subtrees. Note that this doesn’t work every time and that it also makes the computation slower, depending on how many trees your random forest builds.

Important Hyperparameters:

The Hyperparameters in random forest are either used to increase the predictive power of the model or to make the model faster. I will here talk about the hyperparameters of sklearns built-in random forest function.

1. Increasing the Predictive Power

Firstly, there is the **„n_estimators“** hyperparameter, which is just the number of trees the algorithm builds before taking the maximum voting or taking averages of predictions. In general, a higher number of trees increases the performance and makes the predictions more stable, but it also slows down the computation.

Another important hyperparameter is **„max_features“**, which is the maximum number of features Random Forest considers to split a node. Sklearn provides several options, described in their documentation.

The last important hyper-parameter we will talk about in terms of speed, is **„min_sample_leaf “**. This determines, like its name already says, the minimum number of leafs that are required to split an internal node.

2. Increasing the Models Speed

The **„n_jobs“** hyperparameter tells the engine how many processors it is allowed to use. If it has a value of 1, it can only use one processor. A value of “-1” means that there is no limit.

„random_state“ makes the model’s output replicable. The model will always produce the same results when it has a definite value of random_state and if it has been given the same hyperparameters and the same training data.

Lastly, there is the **„oob_score“** (also called oob sampling), which is a random forest cross validation method. In this sampling, about one-third of the data is not used to train the model and can be used to evaluate its performance. These samples are called the out of bag samples. It is very similar to the leave-one-out cross-validation method, but almost no additional computational burden goes along with it.

Advantages and Disadvantages:

Like I already mentioned, an advantage of random forest is that it can be used for both regression and classification tasks and that it’s easy to view the relative importance it assigns to the input features.

Random Forest is also considered as a very handy and easy to use algorithm, because it’s default hyperparameters often produce a good prediction result.

The number of hyperparameters is also not that high and they are straightforward to understand.

One of the big problems in machine learning is overfitting, but most of the time this won't happen that easy to a random forest classifier. That's because if there are enough trees in the forest, the classifier won't overfit the model.

The main limitation of Random Forest is that a large number of trees can make the algorithm to slow and ineffective for real-time predictions. In general, these algorithms are fast to train, but quite slow to create predictions once they are trained. A more accurate prediction requires more trees, which results in a slower model. In most real-world applications the random forest algorithm is fast enough, but there can certainly be situations where run-time performance is important and other approaches would be preferred.

And of course Random Forest is a predictive modeling tool and not a descriptive tool. That means, if you are looking for a description of the relationships in your data, other approaches would be preferred.

Use Cases:

The random forest algorithm is used in a lot of different fields, like Banking, Stock Market, Medicine and E-Commerce. In Banking it is used for example to detect customers who will use the bank's services more frequently than others and repay their debt in time. In this domain it is also used to detect fraud customers who want to scam the bank. In finance, it is used to determine a stock's behaviour in the future. In the healthcare domain it is used to identify the correct combination of components in medicine and to analyze a patient's medical history to identify diseases. And lastly, in E-commerce random forest is used to determine whether a customer will actually like the product or not.

Summary:

Random Forest is a great algorithm to train early in the model development process, to see how it performs and it's hard to build a "bad" Random Forest, because of its simplicity. This algorithm is also a great choice, if you need to develop a model in a short period of time. On top of that, it provides a pretty good indicator of the importance it assigns to your features.

Random Forests are also very hard to beat in terms of performance. Of course you can probably always find a model that can perform better, like a neural network, but these usually take much more time in the development. And on top of that, they can handle a lot of different feature types, like binary, categorical and numerical.

Overall, Random Forest is a (mostly) fast, simple and flexible tool, although it has its limitations.

This post was initially published at my blog (https://machinelearning-blog.com).

Thanks to Ludovic Benistant.

Machine Learning Towards Data Science Random Forest

7.7K claps 32

Niklas Donges Follow

linkedin.com/in/niklas-donges | machinelearning-blog.com

Towards Data Science Follow

Sharing concepts, ideas, and codes.

Responses

Write a response…

Conversation between Marat Kopytjuk and Niklas Donges.

Marat Kopytjuk
Feb 24, 2018

Great post! I would recommend to mention “max_depth” parameter which is crucial for model’s complexity :)

66

1 response

Niklas Donges
Mar 13, 2018

You are right! Thank you.

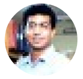
Applause from Niklas Donges (author)

Vivek Solanki
Apr 26, 2018

> *You made it super easy to understand Random forest algorithm..! Thanks a lot for your efforts.* ☺

18

Conversation with Niklas Donges.



Saurabh Prakash

Mar 12, 2018

“When a decision tree generates the nodes and rules, it often uses the information gain and gini index calculations. In comparison, a Random Forest does this randomly.”

The above is not true, RandomForestClassifier takes a parameter “criterion” which can have values “Gini" or entropy.

3

1 response

Niklas Donges

Mar 13, 2018

You are right, thanks for the hint. I edited this sentence.

2

Conversation with Niklas Donges.

Adi
May 2, 2018

Great article! Thank you!

regarding the *min_samples_leaf* parameter, in the documentation it is mentioned that this is the “The minimum number of samples required to be at a leaf node” and not “the number of leafs” as you mentioned. Please clarify.

6

1 response

Niklas Donges
May 5, 2018

I edited the sentence. Thanks for pointing it out.

1

Show all responses