

Ensembles and Random Forests

In this module, you learnt about **ensembles** and understood how they form the basis for another ML model, **random forests**. You also learnt how an ensemble performs better than general ML models. Further, you will learn about different ensemble techniques used in the industry.

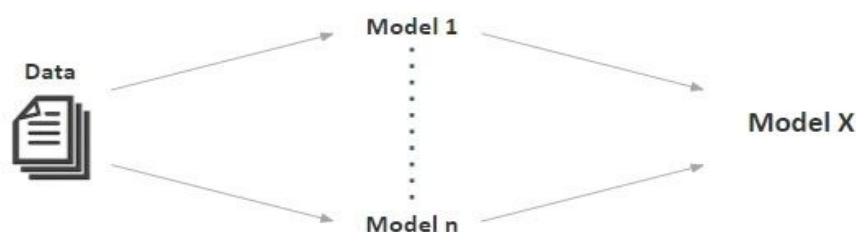
Introduction to Ensembles

A random forest algorithm combines multiple decision trees to generate the final results. This process of combining more than one model to take the final decision is termed as ensemble learning.

An ensemble refers to a group of things viewed as a whole rather than viewing them as individuals. In an ensemble, a collection of models is used to make predictions rather than individual models. Arguably, one of the most popular models in the family of ensemble models is the random forest, which is an ensemble made by a combination of a large number of decision trees. The following are the disadvantages when you stick to a single model to build the final solution:

1. The model may be bound with a specific set of assumptions that the data may or may not follow. For example, if you fit a linear regression model, then you imply that the target variable follows a linear relationship with the attributes. However, this is not always necessary and may lead to less accurate results.
2. Sticking to a single model also implies that the entire data set follows the same trend. If you can identify the variation in relationship with the distribution of different attributes in the data, then you can use multiple models to fine-tune the results. This is partly possible in decision trees, as the data is split based on different attributes; however, the model is presented with other challenges such as high variance and overfitting.

Ensembles attempt to overcome all these challenges by combining different models to predict the final results. These models are considered as the base models, which are combined or aggregated using different techniques to produce the output. In principle, ensembles can be made by combining all types of models. In an ensemble, logistic regression and a few decision trees can work in unison to solve a classification problem. The image below shows how ensemble learning takes place:



Ensemble Learning

Ensemble vs Single Model

Let's understand the limitations faced while following the approach with a single ML model:

1. It has been iterated multiple times that a single model will bound you with its assumptions, and hence, the model may not be as generalizable as it should be.
2. Second, you dealt with limited data to understand the relationship between variables and, finally, replicate it over unseen data. This means that the training data must be explored as extensively as possible to gain a thorough understanding. This activity is restricted if you rely only on a single model.

Now, you are aware of some of the shortcomings of using a single model. However, it is also essential to understand how multiple models come together to solve the problems present in a single model. The model that works extremely well on the training set may not be generalizable, resulting in an overfitted model. On the other hand, if a model is quite generalised, then it will not capture the underlying data, resulting in an underfitted model.

So far, you have learnt about three different algorithms (linear regression, logistic regression and decision trees), and amongst them, decision trees suffer from the problem of high variance and other models suffer to a little extent, which depends on the data set. It happens as you try to obtain the most accurate results with the training set, which leads to overfitting.

If you have a single model to predict the values, it is difficult to predict the exact relationship between variables with a limited set of assumptions. However, in the case of ensembles, multiple models offer the freedom to combine various perspectives to look at data.

The pool of models may consist of the following two types of models: one that captures the underlying pattern in the training data and the other that helps to generalise the results over the unseen data. As a result, the combination of these two can provide a balanced model that performs well on both the training and the test data set.

Choosing Models in Ensemble

Try to recall the two main criteria to choose a model for ensemble learning are diversity and acceptability, which are as follows:

- **Diversity** ensures that the models serve complementary purposes, which means that the individual models make predictions independent of each other, and one model fills the gaps of the other.
- **Acceptability** implies that each model is at least better than a random model. This is a lenient criterion for each model to be accepted into the ensemble, which is that it has to be at least better than a random guesser.

You can bring diversity among the base models that you plan to include in your ensemble in several ways, which are as follows:

- Use different subsets of training data
- Use different training hyperparameters
- Use different classes of models

- Use different features sets for each of the models

Experimenting with an Ensemble of Three Models

To understand why ensembles work better than individual models, let's take a simple example of three biased coins (models). Consider an ensemble of three models: m1, m2 and m3, for a binary classification task (say, 1 or 0). Suppose each of these models has a probability of being correct 70% of the time.

So, each model is acceptable. Given a data point whose class has to be predicted, the ensemble will predict the class using a majority score. In other words, if two or more models predict class = 1 as the output, the ensemble will predict 1, and vice versa.

The following table shows all the possible cases that can occur while classifying a test data point as 1 or 0. The column to the extreme right shows the probability of each case. For example, if you take the first row, all m1, m2, and m3 give the correct output. Hence, the probability for this case becomes $(0.7 \times 0.7 \times 0.7)$, since each model has a probability of being correct 70% of the time and all three models are completely independent of each other. Similarly, the probability of the second row becomes $(0.7 \times 0.7 \times 0.3)$ since you have "Correct", "Correct", and "Incorrect". And so on for all the rows.

Case	Result of Each Model			Result of the Ensemble	Probability
	m1	m2	m3		
1	Correct	Correct	Correct	Correct	$0.7 \times 0.7 \times 0.7 = 0.343$
2	Correct	Correct	Incorrect	Correct	$0.7 \times 0.7 \times 0.3 = 0.147$
3	Correct	Incorrect	Correct	Correct	$0.7 \times 0.3 \times 0.7 = 0.147$
4	Incorrect	Correct	Correct	Correct	$0.3 \times 0.7 \times 0.7 = 0.147$
5	Incorrect	Incorrect	Correct	Incorrect	$0.3 \times 0.3 \times 0.7 = 0.063$
6	Incorrect	Correct	Incorrect	Incorrect	$0.3 \times 0.7 \times 0.3 = 0.063$
7	Correct	Incorrect	Incorrect	Incorrect	$0.7 \times 0.3 \times 0.3 = 0.063$
8	Incorrect	Incorrect	Incorrect	Incorrect	$0.3 \times 0.3 \times 0.3 = 0.027$

Ensemble Model

In this table, there are four cases each where the decision of the final model (ensemble) is either correct or incorrect. Let's assume that the probability of the ensemble being correct is p , and the probability of the ensemble being incorrect is q .

For the data in the table, p and q can be calculated as follows:

$$p = 0.343 + 0.147 + 0.147 + 0.147 = 0.784$$

$$q = 0.027 + 0.063 + 0.063 + 0.063 = 0.216 = 1 - p$$

Notice how the ensemble has a higher probability of being correct and a lower probability of being incorrect than any of the individual models ($0.78 > 0.70$ and $0.216 < 0.30$). In this way, you can also calculate the probabilities of the ensemble being correct and incorrect with 4, 5, 100, 1000, and even a million individual models. The difference in probabilities will increase with an increasing number of models, thus improving the overall performance of the ensemble.

Ensemble Techniques

Some of the common ensemble techniques are as follows:

- Voting
- Stacking
- Blending
- Boosting
- Bagging

Voting combines the output of different algorithms by means of a vote. In the case of a classification problem, the output of the model will be the class predicted by the majority of base classifiers. In the case of a regression problem, it will be the average of all the predictions made by the individual models. In this way, every classifier or regressor will have an equal weightage in the final prediction. However, the weights allocated to the different base models may be varied in order to generate the final results.

A base model that performs better than the other models can be provided with a higher weightage in decision-making. This is the high-level approach followed in **stacking** and **blending**. The outputs of the base models are passed through a level-2 classifier or regressor. The performance of the final model can be enhanced further by running the base models through another classifier/regressor. This will help you to put a weight on each base model for final prediction based on their performance. A stronger base model will have a higher weightage than a weaker base model. In this way, the individual models are combined with different weights to obtain the final prediction. However, this should be used with a check as it can lead to overfitting.

Boosting is one of the most popular ensemble techniques. It can be used with any algorithm, as it generates weak learners sequentially to create an ensemble of weak learners, which, in turn, has a good performance.

Bagging exploits the 'diversity' feature in ensembles. Bagging works well with the algorithms that are unstable and result in a high variation with a few changes in the data, that is, models with a high variance. Try to recall that decision trees tend to suffer from this problem if the hyperparameters are not tuned properly. Hence, bagging works quite well for high-variance models such as decision trees. Random forests are built on this approach with some additional improvements and are powerful at reducing the variance of an algorithm.

However, this technique is associated with some disadvantages. With this approach, you cannot explore or justify individual models, as the data is selected randomly for each subset. This leads to a loss of interpretability. Moreover, as multiple models are built simultaneously, bagging can be computationally expensive and is used on a case-to-case basis.

Introduction to Random Forests

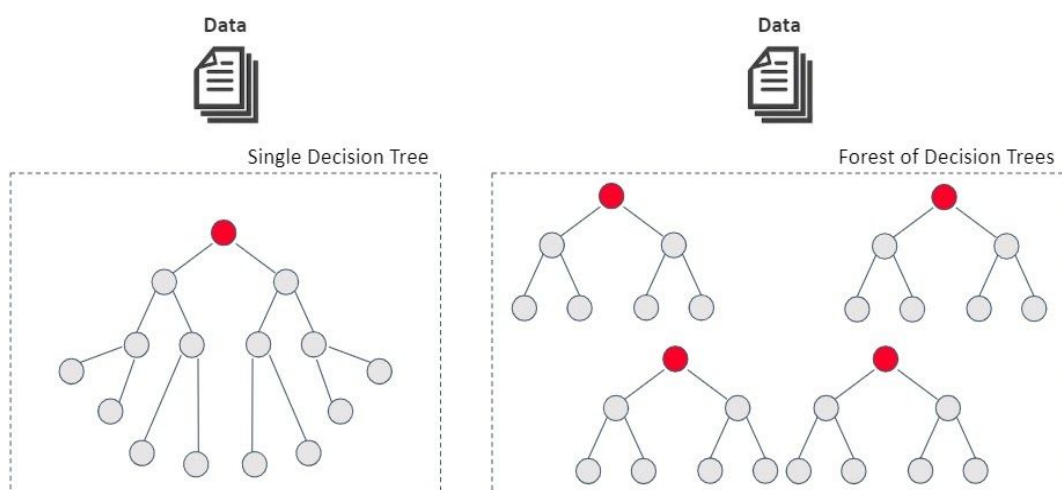
The random forest algorithm is an ensemble of decision trees that uses bagging to generate different base models. So far, random forest has been the most successful model among the bagging ensembles. They are essentially ensembles of a number of decision trees. You can create a large number of models (say, 100 decision trees), each one on a different bootstrap sample from the training set. To get the result, you can aggregate the decisions taken by all the trees in the ensemble. Aggregation combines the results of different models present in the ensemble.

Bootstrapping refers to creating bootstrap samples from a given data set. A bootstrap sample is created by sampling the given data set uniformly and with replacement. This means that different subsets have overlapping data points. A bootstrap sample typically contains approximately 40–70% data from the data set. The base models in the algorithm are generated by implementing all the steps mentioned under decision trees on each subset. You must understand that bagging is a technique in itself and is not specific to random forests.

A random forest selects a random sample of data points (bootstrap sample) to build each tree and a random sample of features while splitting a node. Randomly selecting features ensures that each tree is diverse and that some prominent features are dominating in all the trees making them somewhat similar.

In the random forest algorithm, you end up with different training subsets with overlapping data points. Hence, you have an overlapping testing set for each base model as well. Therefore, the algorithm aggregates the results of each model on every point.

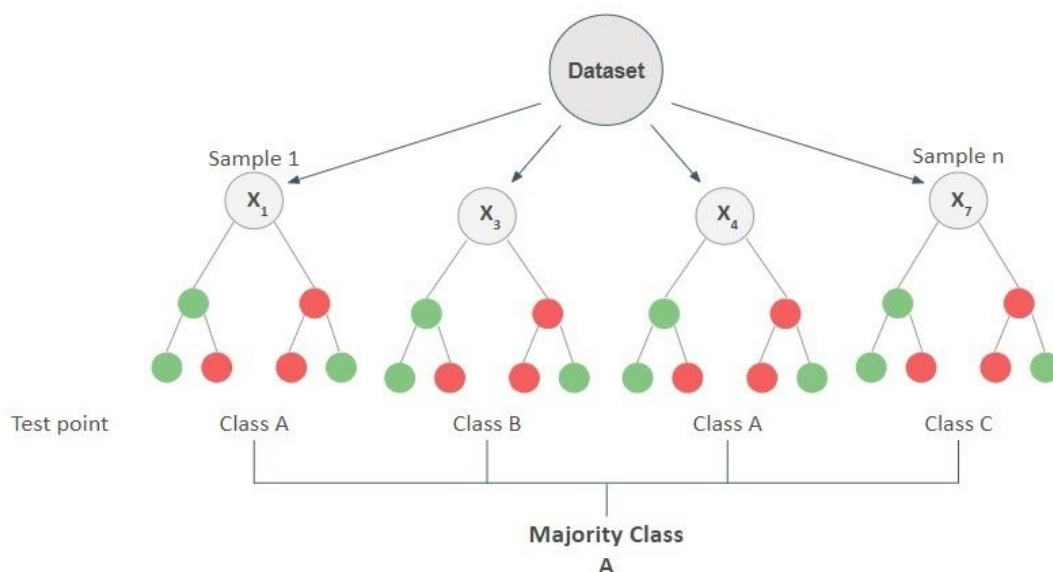
In case of a classification problem it takes a majority vote on the predictions made by models for every data point and provides the output as the class that has received the highest number of votes. In the case of regression, it will be the average of all the predicted values available for a particular data point.



Random Forest: Ensemble of Decision Trees

Consider a random forest of 10 decision trees. A summary of the steps of the algorithm is as follows:

- First, the algorithm will generate 10 bootstrapped samples from the sample data.
- Next, each sample will be used to train a decision tree. Remember that the set of features used to split at each node of every tree changes and is randomly selected. In this way, you create 10 decision trees.
- Recall that in a decision tree, every data point passes from the root node to the bottom until it is classified in a leaf node. A similar process occurs in random forests while making predictions. Each data point passes through different trees in the ensemble that is built on different training and feature subsets.
- Then, the final outcomes of each tree are combined either by taking the most frequent class prediction in the case of a classification problem or by taking an average in the case of a regression problem.

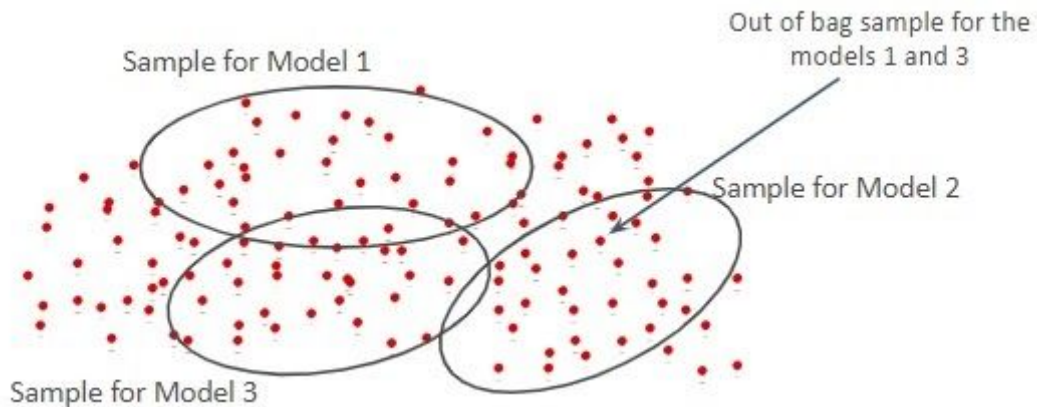


Random Forest Algorithm

Model Evaluation: Out-of-Bag Score

Generally, for evaluating a machine learning model, you split the data set into a training and test data. However, the random forest algorithm can be used without splitting as well. The base models in the algorithm are built on a subset of the training data, and hence, the entire data is automatically divided into the following two parts: a training set and a validation set. Thus, it omits the need for a set-aside test data. However, you can still keep part of the data as the testing set to test the model on unseen data.

The validation set consists of all the data points that were not used by the base model to train the tree. These are termed as **out-of-Bag (OOB)** samples for the individual tree. Every tree has a separate validation set as samples are bootstrapped for each tree.



Out-of-Bag Samples

The OOB error can be calculated as the count of incorrect predictions by the proportion of the total number of predictions on out-of-bag (OOB) samples.

$$\text{Out of bag error} = \text{Number of incorrectly predicted values (out of bag)} \div \text{Total data points}$$

You know that the OOB error is beneficial when your data set is small. Each observation of the training set is used as a test observation for OOB score. Since each tree is built on a bootstrap sample, each observation can be used as a test observation by those trees which did not have it in their bootstrap sample. All these trees predict on this observation and you get an error for a single observation. The final OOB error is calculated by calculating the error on each observation and aggregating it.

It turns out that the OOB error is as good as a cross validation error. OOB score in random forests is similar to the cross-validation score. OOB score is a technique to measure the prediction error of random forests. It gives a similar estimate to the one produced by the cross-validation score.

One of the factors that could affect the performance of a random forest model is:

- **Correlation between trees in the forest**

As the model is based on ensembles, one of the main requirements of the model is diversity. Correlated base models means that the ensemble lacks diversity and, hence, cannot perform better than the individual models.

Time taken to build a forest

To construct a forest of S trees, on a dataset which has M features and N observations, the time taken will depend on the following factors:

1. The number of trees. The time is directly proportional to the number of trees. But this time can be reduced by creating the trees in parallel.
2. The size of the bootstrap sample. Generally the size of a bootstrap sample is 30-70% of N . The smaller the size the faster it takes to create a forest.

3. The size of subset of features while splitting a node. Generally this is taken as \sqrt{M} in classification and $M/3$ in regression.

Advantages, Disadvantages and Applications

In this segment, you learnt about the advantages and disadvantages of the random forest model. The advantages are as follows:

1. **Diversity**
Diversity arises because each tree is created with a subset of the attributes/features/variables, i.e., not all the attributes are considered while making each tree; the choice of the attributes is random. This ensures that the trees are independent of each other.
2. **Stability**
Stability arises because the answers given by a large number of trees average out. A random forest has a lower model variance than an ordinary individual tree.
3. **Immunity to the curse of dimensionality**
Since each tree does not consider all the features, the feature space (the number of features that a model has to consider) reduces. This makes an algorithm immune to the curse of dimensionality. Also, a large feature space causes computational and complexity issues.
4. **Parallelisation**
You need a number of trees to make a forest. Since two trees are independently built on different data and attributes, they can be built separately. This implies that you can make full use of your multi-core CPU to build random forests. Suppose there are 4 cores and 100 trees to be built; each core can build 25 trees to make a forest.

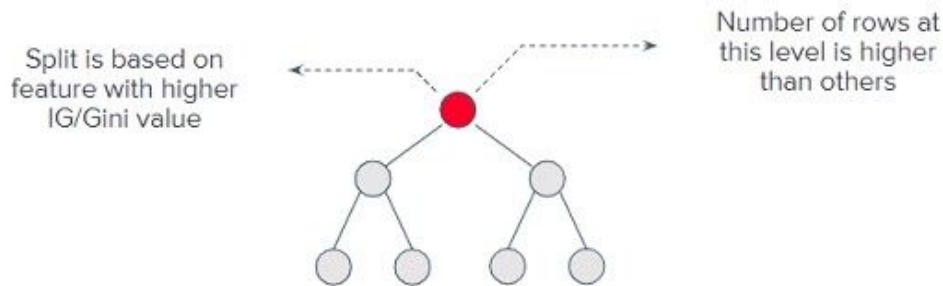
The disadvantages of the random forest model are as follows:

1. **Lack of interpretability**
They are less intuitive and interpretable as compared to a single decision tree as many trees aggregate together to produce the final result in a random forest.
2. **High computational costs**
Random forests are much complex, time consuming and computationally expensive.

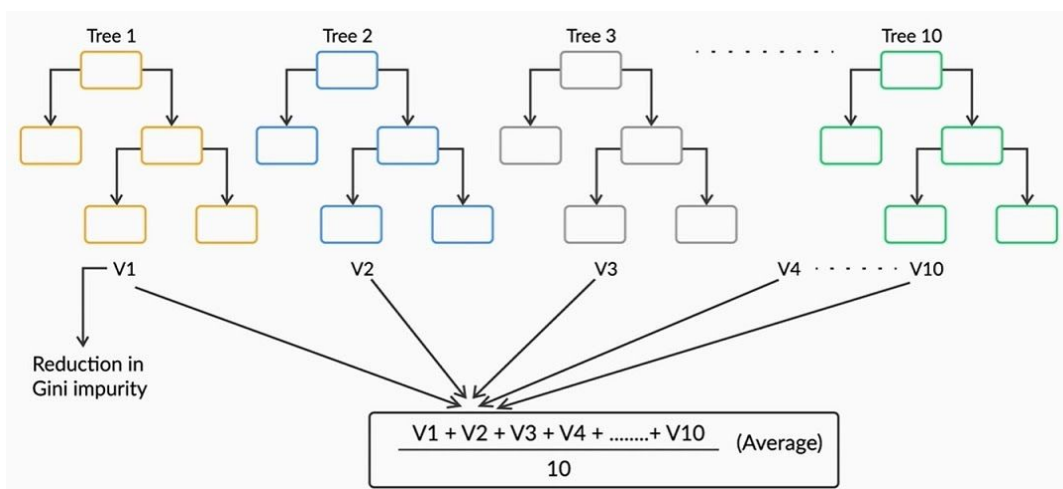
Feature Importance in Random Forests

Feature importance plays a crucial role in contributing towards effective prediction, decision-making and model performance. It eliminates the less critical variables from a large data set and helps in identifying the key features that can lead to better prediction results.

Random forests use multiple trees, reduce variance and allow for further exploration of feature combinations. The importance of features in random forests, sometimes called **Gini importance** or **mean decrease impurity**, is defined as the **total decrease in node impurity**. It is calculated by taking a weighted average of the metric mentioned above across all the trees in the ensemble. This is replicated for all the features one-by-one.



The weights associated with a feature for every tree can be simply calculated as the **fraction of rows** present in the node where it will split the data set. The number of rows provides an approximation for the importance of the feature, as the node at a higher level will have more number of rows in them.



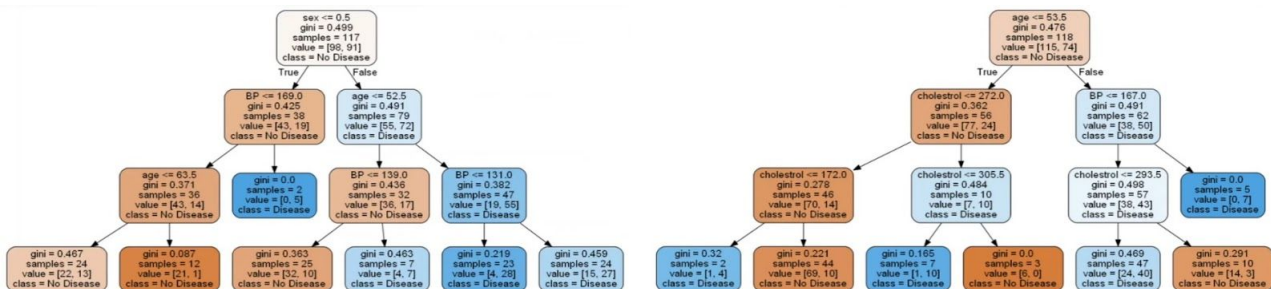
For each variable, the sum of the information gain across every tree of the forest is accumulated every time a variable is chosen to split a node. The value is then divided by the number of base trees in which the feature is involved to give the final average value.

Python: Heart Disease Prediction

In this segment, you learnt how to implement random forests in Python using the sklearn library. You worked with the heart disease data to build a random forest model to predict whether a person has heart disease or not. The data lists the results of various tests that were conducted on patients along with some other details of the patients.

- Heart disease = 0 means that the person does not have any heart disease.
- Heart disease = 1 means that the person has a heart disease.
- sex = 0 means that the person is female.
- sex = 1 means that the person is male.

You can easily build a random forest model in Python using the `RandomForestClassifier()`. As part of this process, you must have learnt about some sample trees and understood how the process takes place internally. The two sample trees used in the process are presented in the image given below:



You also learnt how to calculate the OOB score generated by the classifier to understand how the collection of these individual trees perform. Hyperparameter tuning can result in drastic improvements to the model. The attribute 'age' contributes the maximum to the decision of whether a person has a heart disease or not.

	Varname	Imp
0	age	0.375397
3	cholesterol	0.278449
2	BP	0.208346
1	sex	0.137808

To summarise, you learnt how to build a random forest in sklearn. You learnt about the following estimators apart from the ones you learnt in the decision tree module:

- **max_features**
This feature helps you decide the number of attributes that the algorithm will consider when the splitting criteria is checked.
- **n_estimators**
This defines the number of decision trees that you will have in the random forest.

Python: Housing Price Prediction

In this segment, you learnt how to run a regression model using the random forest. For this, you used the housing data set to predict house prices based on various factors such as the area, the number of bedrooms and parking space that you already learnt about in the previous modules. Essentially, the aim was to:

- Know the variables that significantly contribute to predicting house prices,
- Create a linear model that quantitatively relates house prices with variables, such as the number of rooms, the area and the number of bathrooms, and
- Develop a random forest regressor to predict the house prices.

First, you converted the categorical variables into numerical indices using dummy variables. Next, you split the data in a ratio of 70:30. Finally, you can use the `RandomForestRegressor()` to build the model. This model gave good results on the training and the testing data.

Model Comparison: Telecom Churn Prediction

In this segment, you learnt how decision trees and random forests stack up against logistic regression. You used the telecom churn prediction example that was considered in the logistic regression module. The problem statement was as follows:

You have a telecom firm that collected data of all its customers. The main types of attributes are as follows:

- Demographics (age, gender, etc.)
- Services availed (internet packs purchased, special offers taken, etc.)
- Expenses (amount of recharge done per month, etc.)

Based on all this past information, you want to build a model that will predict whether a particular customer will churn or not, i.e., whether they will switch to a different service provider or not. So, the variable of interest, i.e., the target variable here is 'Churn', which will tell us whether or not a particular customer has churned. It is a binary variable, where 1 means that the customer has churned and 0 means that the customer has not churned.

You can recall that without putting much effort in scaling, multicollinearity, p-values and feature selection, you obtained impressive and better results using decision trees than those obtained using a logistic regression model. However, remember that decision trees are high-variance models and that they change quite rapidly with small changes in the training data. In such a case, you either prune the tree to reduce variance or use an ensemble method such as the random forest method.

Random forests definitely result in a great improvement in the results compared with logistic regression and decision trees with much less effort. It has exploited the predictive power of decision trees and learnt much more than a single decision tree could learn alone. However, there is not much visibility with respect to the key features and the direction of their effects on the prediction, which is done well by a logistic regression model. If interpretation is not of key significance, then random forests definitely do a great job.

Disclaimer: All content and material on the upGrad website is copyrighted material, either belonging to upGrad or its bonafide contributors and is purely for the dissemination of education. You are permitted to access, print and download extracts from this site purely for your own education only and on the following basis:

- You can download this document from the website for self-use only.
- Any copy of this document, in part or full, saved to disk or to any other storage medium, may only be used for subsequent, self-viewing purposes or to print an individual extract or copy for non-commercial personal use only.
- Any further dissemination, distribution, reproduction, copying of the content of the document herein or the uploading thereof on other websites or use of the content for any other commercial/unauthorised purposes in any way which could infringe the intellectual property rights of upGrad or its contributors, is strictly prohibited.
- No graphics, images or photographs from any accompanying text in this document will be used separately for unauthorised purposes.
- No material in this document will be modified, adapted or altered in any way.
- No part of this document or upGrad content may be reproduced or stored in any other website or included in any public or private electronic retrieval system or service without upGrad's prior written permission.
- Any right not expressly granted in these terms are reserved.



Introduction to Random Forest in Machine Learning

December 11, 2020

Topics: [Machine Learning](#)

A random forest is a supervised machine learning algorithm that is constructed from decision tree algorithms. This algorithm is applied in various industries such as banking and e-commerce to predict behavior and outcomes.

This article provides an overview of the random forest algorithm and how it works. The article will present the algorithm's features and how



A random forest is a machine learning technique that's used to solve regression and classification problems. It utilizes ensemble learning, which is a technique that combines many classifiers to provide solutions to complex problems.

A random forest algorithm consists of many decision trees. The 'forest' generated by the random forest algorithm is trained through bagging or bootstrap aggregating. Bagging is an ensemble meta-algorithm that improves the accuracy of machine learning algorithms.

The (random forest) algorithm establishes the outcome based on the predictions of the decision trees. It predicts by taking the average or mean of the output from various trees. Increasing the number of trees increases the precision of the outcome.

A random forest eradicates the limitations of a decision tree algorithm. It reduces the overfitting of datasets and increases precision. It generates predictions without requiring many configurations in packages (like [scikit-learn](#)).

Features of a Random Forest Algorithm

It's more accurate than the decision tree algorithm.

It provides an effective way of handling missing data.

It can produce a reasonable prediction without hyper-parameter tuning.

How random forest algorithm works

Understanding decision trees

Decision trees are the building blocks of a random forest algorithm. A decision tree is a decision support technique that forms a tree-like structure. An overview of decision trees will help us understand how random forest algorithms work.

A decision tree consists of three components: decision nodes, leaf nodes, and a root node. A decision tree algorithm divides a training dataset into branches, which further segregate into other branches. This sequence continues until a leaf node is attained. The leaf node cannot be segregated further.

The nodes in the decision tree represent attributes that are used for predicting the outcome. Decision nodes provide a link to the leaves. The following diagram shows the three types of nodes in a decision tree.

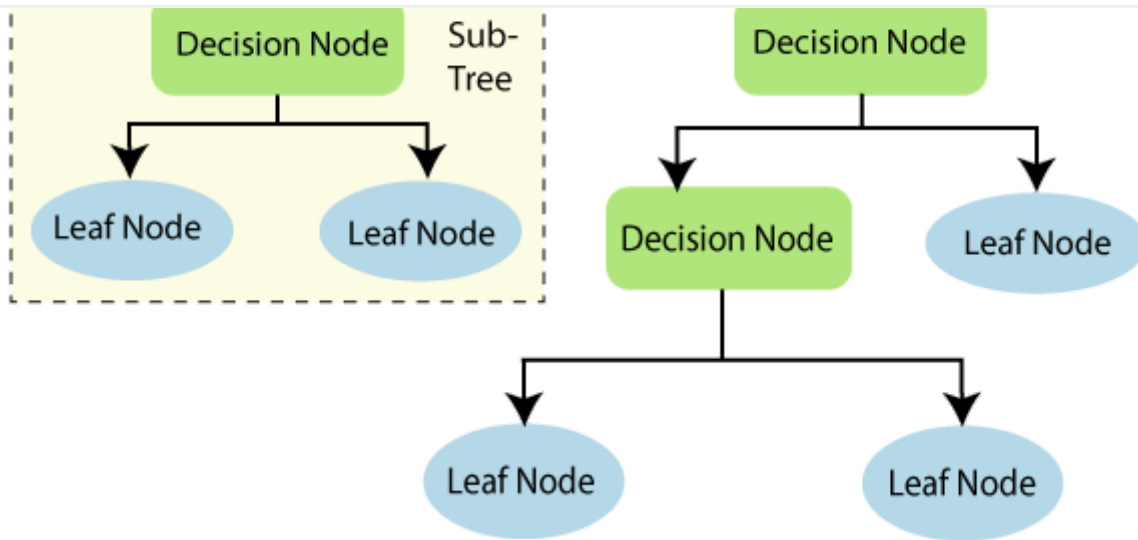


Image Source: Tutorials and Example

The information theory can provide more information on how decision trees work. Entropy and information gain are the building blocks of decision trees. An overview of these fundamental concepts will improve our understanding of how decision trees are built.

Entropy is a metric for calculating uncertainty. Information gain is a measure of how uncertainty in the target variable is reduced, given a set of independent variables.

The information gain concept involves using independent variables (features) to gain information about a target variable (class). The entropy of the target variable (Y) and the **conditional entropy** of Y (given X) are used to estimate the information gain. In this case, the conditional entropy is subtracted from the entropy of Y.



is an important activity in the construction of decision trees.

Let's take a simple example of how a decision tree works. Suppose we want to predict if a customer will purchase a mobile phone or not. The features of the phone form the basis of his decision. This analysis can be presented in a decision tree diagram.

The root node and decision nodes of the decision represent the features of the phone mentioned above. The leaf node represents the final output, either *buying* or *not buying*. The main features that determine the choice include the price, internal storage, and Random Access Memory (RAM). The decision tree will appear as follows.

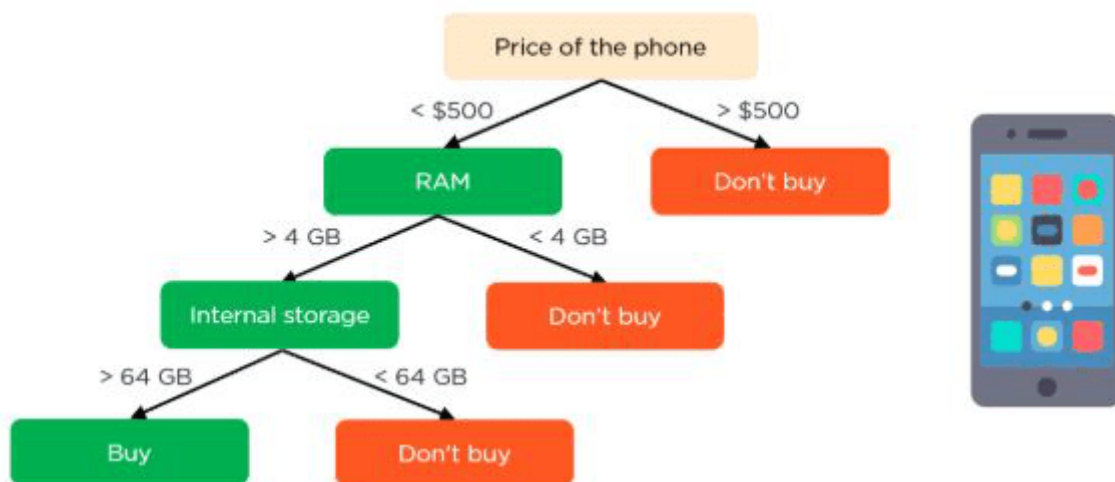


Image Source: Simplilearn

Applying decision trees in random forest

Bagging involves using different samples of data (training data) rather than just one sample. A training dataset comprises observations and features that are used for making predictions. The decision trees produce different outputs, depending on the training data fed to the random forest algorithm. These outputs will be ranked, and the highest will be selected as the final output.

Our first example can still be used to explain how random forests work. Instead of having a single decision tree, the random forest will have many decision trees. Let's assume we have only four decision trees. In this case, the training data comprising the phone's observations and features will be divided into four root nodes.

The root nodes could represent four features that could influence the customer's choice (price, internal storage, camera, and RAM). The random forest will split the nodes by selecting features randomly. The final prediction will be selected based on the outcome of the four trees.

The outcome chosen by most decision trees will be the final choice. If three trees predict *buying*, and one tree predicts *not buying*, then the final prediction will be *buying*. In this case, it's predicted that the customer will buy the phone.

Classification in random forests

Classification in random forests employs an ensemble methodology to attain the outcome. The training data is fed to train various decision

consists of decision nodes, leaf nodes, and a root node. The leaf node of each tree is the final output produced by that specific decision tree. The selection of the final output follows the majority-voting system. In this case, the output chosen by the majority of the decision trees becomes the final output of the random forest system. The diagram below shows a simple random forest classifier.

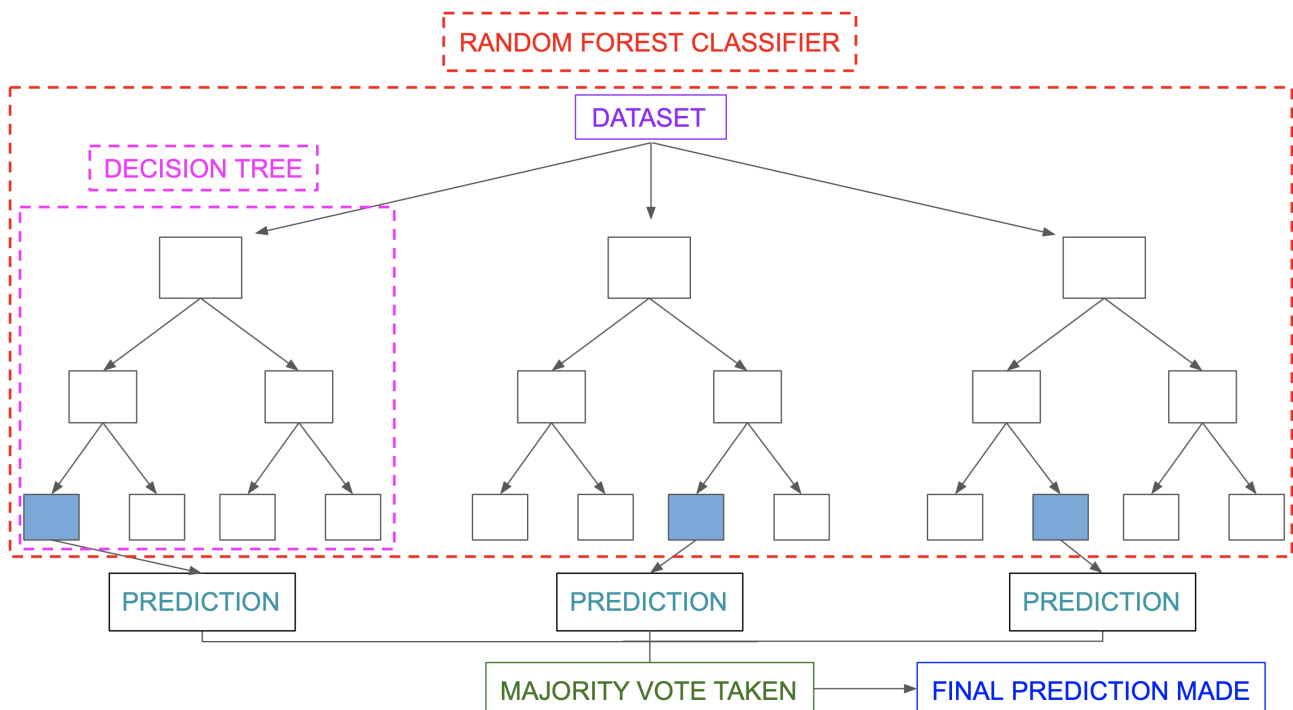


Image Source: Medium

Let's take an example of a training dataset consisting of various fruits such as bananas, apples, pineapples, and mangoes. The random forest classifier divides this dataset into subsets. These subsets are given to every decision tree in the random forest system. Each decision tree

random forest classifier collects the majority voting to provide the final prediction. The majority of the decision trees have chosen *apple* as their prediction. This makes the classifier choose *apple* as the final prediction.

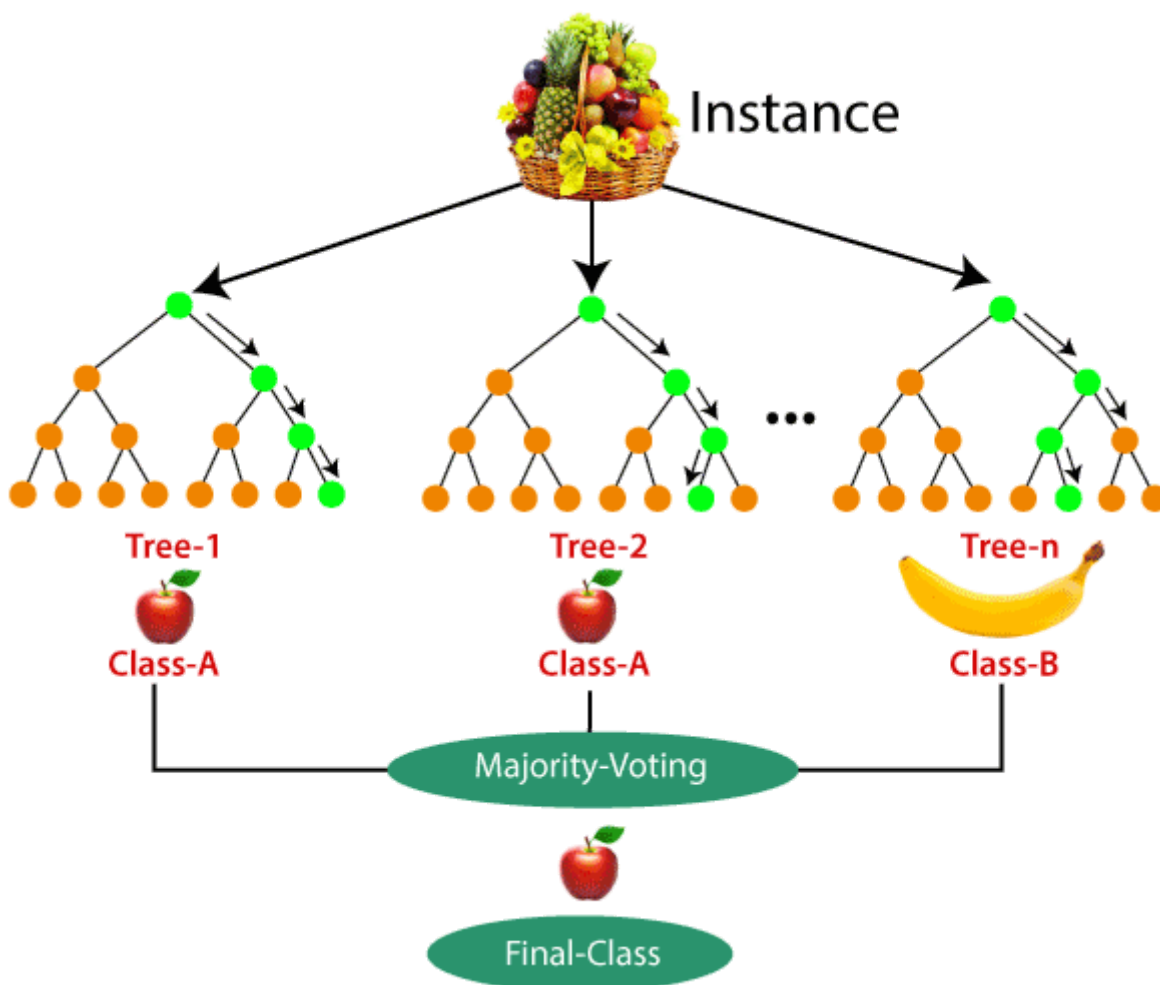


Image Source: Javatpoint

Regression in random forests

We can run random forest regressions in various programs such as [SAS](#), R, and python. In a random forest regression, each tree produces a specific prediction. The mean prediction of the individual trees is the output of the regression. This is contrary to random forest classification, whose output is determined by the mode of the decision trees' class.

Although random forest regression and linear regression follow the same concept, they differ in terms of functions. The function of linear regression is $y = bx + c$, where y is the dependent variable, x is the independent variable, b is the estimation parameter, and c is a constant. The function of a complex random forest regression is like a [blackbox](#).

Applications of random forest

Some of the applications of the random forest may include:

Banking

Random forest is used in banking to predict the creditworthiness of a loan applicant. This helps the lending institution make a good decision on whether to give the customer the loan or not. Banks also use the random forest algorithm to detect fraudsters.

Health care

Stock market

Financial analysts use it to identify potential markets for stocks. It also enables them to identify the behavior of stocks.

E-commerce

Through rain forest algorithms, e-commerce vendors can predict the preference of customers based on past consumption behavior.

When to avoid using random forests

Random forest algorithms are not ideal in the following situations:

Extrapolation

Random forest regression is not ideal in the [extrapolation](#) of data. Unlike linear regression, which uses existing observations to estimate values beyond the observation range. This explains why most applications of random forest relate to classification.

Sparse data

Random forest does not produce good results when the data is very sparse. In this case, the subset of features and the bootstrapped sample will produce an invariant space. This will lead to unproductive splits, which will affect the outcome.



A random forest produces good predictions that can be understood easily.

It can handle large datasets efficiently.

The random forest algorithm provides a higher level of accuracy in predicting outcomes over the decision tree algorithm.

Disadvantages of random forest

When using a random forest, more resources are required for computation.

It consumes more time compared to a decision tree algorithm.

Conclusion

The rain forest algorithm is a machine learning algorithm that is easy to use and flexible. It uses ensemble learning, which enables organizations to solve regression and classification problems.

This is an ideal algorithm for developers because it solves the problem of overfitting of datasets. It's a very resourceful tool for making accurate predictions needed in strategic decision making in organizations.

Resources

[Dev Skrol](#)

Peer Review Contributions by: [Lalithnarayan C](#)

About the author



Onesmus Mbaabu

Onesmus Mbaabu is a Ph.D. candidate pursuing a doctoral degree in Management Science and Engineering at the School of Management and Economics, University of Electronic Science and Technology of China (UESTC), Sichuan Province, China.

His interests include economics, data science, emerging technologies, and information systems. His hobbies are playing basketball and listening to music.

This article was contributed by a student member of Section's Engineering Education Program. Please report any errors or inaccuracies to enged@section.io.

SOLVING THE EDGE PUZZLE: Your journey to the Edge begins here.

[Read White Paper](#)

Discover Section's community-generated pool of resources from the next generation of engineers.

[Learn more](#)[+ QUICK LINKS // More Section offerings](#)

Join our Slack community

[Add to Slack](#)

Company

[About](#)[Careers](#)[Legals](#)

Resources

[Blog](#)

SOLVING THE EDGE PUZZLE: Your journey to the Edge begins here.



Read White Paper





[Partners](#)[Changelog](#)

Support

[Docs](#)[Community Slack](#)[Help & Support](#)[Platform Status](#)

Section supports many open source projects including:

 [varnish cache
logo](#) [cloud native
computing foundation
logo](#) [the linux
foundation logo](#) [lf edge
logo](#)

© 2021 Section

[Privacy Policy](#) [Terms of Service](#)

More than one instance of Sumo is attempting to start on this page. Please check that you are only loading Sumo once per page.

NATIONALLY
LOCALLY
REMOTELY

FOR EMPLOYERS



A Complete Guide to the Random Forest Algorithm

All you need to know about the random forest model.

Niklas Donges

July 22, 2021

Updated: September 16, 2021

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 of its simplicity and diversity (it can be used for both classification and regression tasks). In this post we'll learn how the random forest algorithm works, how it differs from other algorithms and how to use it.

WHAT IS RANDOM FOREST?

Random forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, usually trained with the "bagging" method. The general idea of the bagging method is that a combination of learning models increases the overall result.

TABLE OF CONTENTS

How it works

View remote jobs at top tech companies nationwide

BETA

NATIONALLY
LOCALLY
REMOTELY

FOR EMPLOYERS

- [Advantages and disadvantages of the random forest algorithm](#)
- [Random Forest use cases](#)
- [Summary](#)

HIRING NOW

[View All Data Science Jobs](#)

How Random Forest Works

Random forest is a [supervised learning algorithm](#). The "forest" it builds, is an ensemble of decision trees, usually trained with the "bagging" method. The general idea of the [bagging method](#) is that a combination of learning models increases the overall result.

Put simply: random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

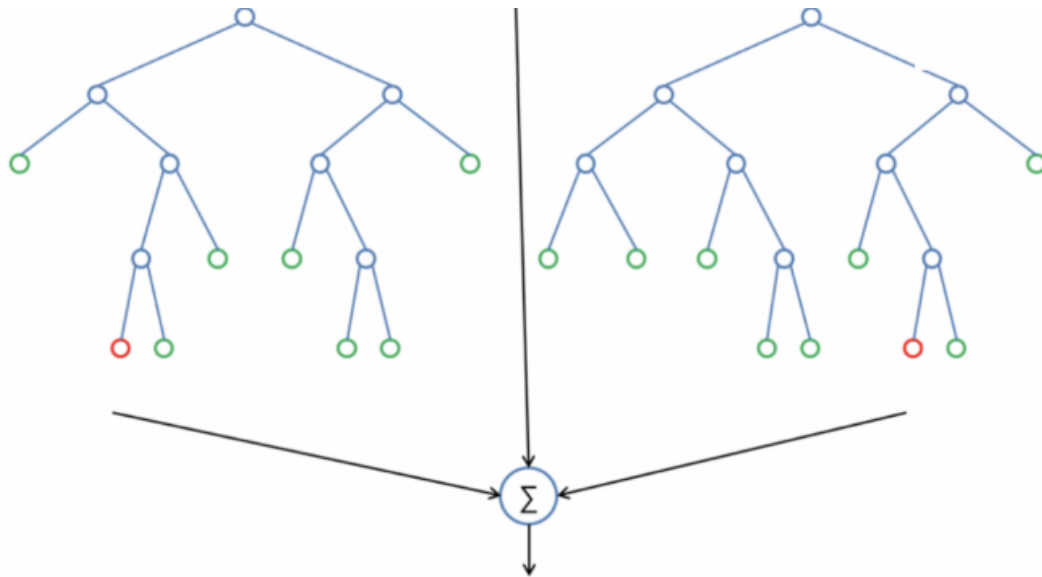
One big advantage of random forest is that it can be used for both classification and regression problems, which form the majority of current machine learning systems. Let's look at random forest in classification, since classification is sometimes considered the building block of machine learning. Below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

[View remote jobs at top tech companies nationwide](#)

BETA

NATIONALLY
LOCALLY
REMOTELY

FOR EMPLOYERS



Random forest has nearly the same hyperparameters as a decision tree or a bagging classifier. Fortunately, there's no need to combine a decision tree with a bagging classifier because you can easily use the classifier-class of random forest. With random forest, you can also deal with regression tasks by using the algorithm's 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).

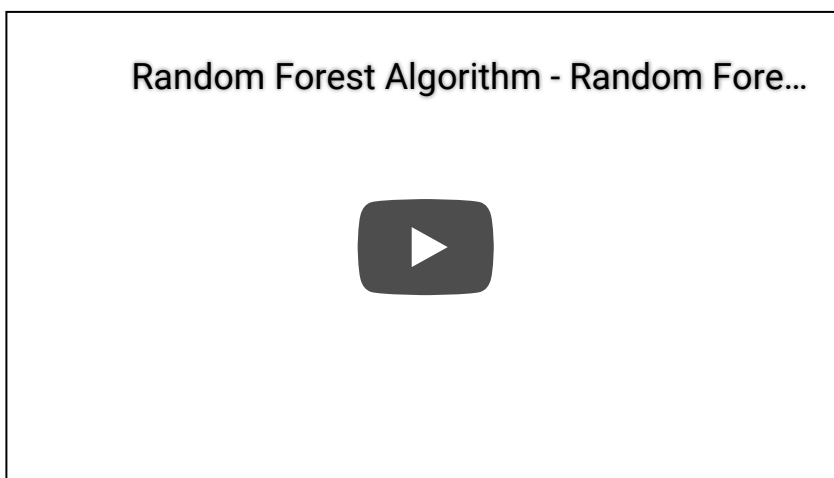
Real-Life Analogy

Andrew wants to decide where to go during one-year vacation, so he asks the

View remote jobs at top tech companies nationwide

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

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



This in-depth tutorial will help you to even better understand random forest algorithms

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 feature's importance by looking at how much the tree nodes that 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 the sum of all importance is equal to one.

τC	1	1	1	1	1	0	0	0	1	1	1	1	0	1	0	1	0
----------	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

View remote jobs at top tech companies nationwide

BETA

NATIONALLY
LOCALLY
REMOTELY

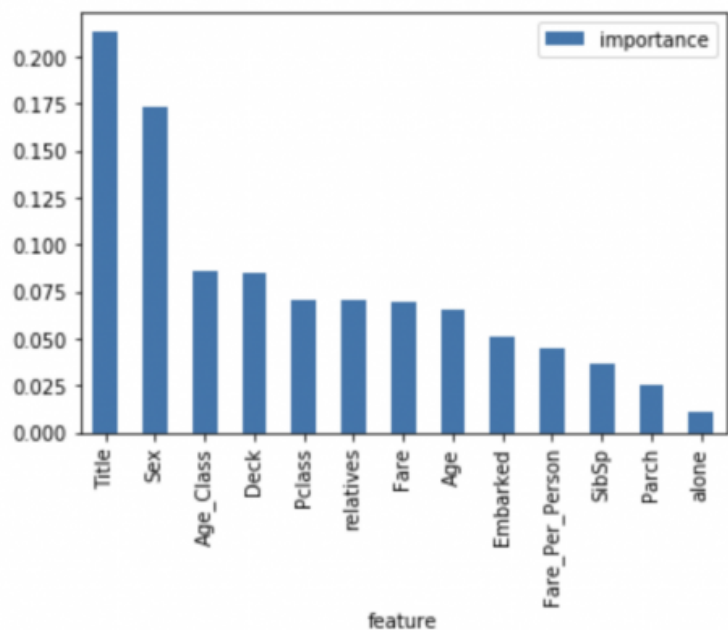
FOR EMPLOYERS

children is a leaf."

By looking at the feature importance you can decide which features to possibly drop because they don't contribute enough (or sometimes nothing at all) 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 is a table and visualization showing 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](#).

feature	importance
Title	0.213
Sex	0.173
Age_Class	0.086
Deck	0.085
Pclass	0.071
relatives	0.070
Fare	0.069
Age	0.065
Embarked	0.051
Fare_Per_Person	0.045
SibSp	0.037
Parch	0.025
alone	0.011



View remote jobs at top tech companies nationwide

BETA



FOR EMPLOYERS

formulate some set of rules, which will be used to make the predictions.

For example, to predict whether a person will click on an online advertisement, you might collect the ads the person clicked on in the past and some features that describe his/her decision. If you put the features and labels into a decision tree, it will generate some rules that help 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 "deep" decision trees might suffer from overfitting. Most of the time, random forest prevents this by creating random subsets of the features and building smaller trees using those subsets. Afterwards, it combines the subtrees. It's important to note this doesn't work every time and it also makes the computation slower, depending on how many trees the 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. Let's look at the hyperparameters of sklearn's 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 the 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, all described in the [documentation](#).

View remote jobs at top tech companies nationwide

BETA

NATIONALLY
LOCALLY
REMOTELY

FOR EMPLOYERS

use. If it has a value of one, it can only use one processor. A value of “-1” means that there is no limit.

The **random_state** hyperparameter 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's very similar to the leave-one-out-cross-validation method, but almost no additional computational burden goes along with it.

See all **Data + Analytics** jobs at top tech companies & startups

[VIEW JOBS](#)

Advantages and Disadvantages of the Random Forest Algorithm

One of the biggest advantages of random forest is its versatility. It can be used for both regression and classification tasks, and it’s also easy to view the relative importance it assigns to the input features.

Random forest is also a very handy algorithm because the default hyperparameters it uses often produce a good prediction result. Understanding the

[View remote jobs at top tech companies nationwide](#)

BETA



FOR EMPLOYERS

One of the biggest problems in machine learning is overfitting, but most of the time this won't happen thanks to the random forest classifier. If there are enough trees in the forest, the classifier won't overfit the model.

WHAT IS A RANDOM FOREST CLASSIFIER?

The term "Random Forest Classifier" refers to the classification algorithm made up of several decision trees. The algorithm uses randomness to build each individual tree to promote uncorrelated forests, which then uses the forest's predictive powers to make accurate decisions.

The main limitation of random forest is that a large number of trees can make the algorithm too 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, meaning if you're looking for a description of the relationships in your data, other approaches would be better.

Random Forest Use Cases

The random forest algorithm is used in a lot of different fields, like banking, the stock market, medicine and e-commerce.

In finance, for example, it is used to detect customers more likely to repay their debt on time, or use a bank's services more frequently. In this domain it is also used to detect fraudsters out to scam the bank. In trading, the algorithm can be used to determine a stock's future behavior.

[View remote jobs at top tech companies nationwide](https://builtin.com/data-science/random-forest-algorithm)

BETA

NATIONALLY
LOCALLY
REMOTELY

FOR EMPLOYERS

Random forest is used in e-commerce 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. Its simplicity makes building a “bad” random forest a tough proposition.

The algorithm is also a great choice for anyone who needs to develop a model quickly. 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 performance wise. Of course, you can probably always find a model that can perform better, like a neural network for example, but these usually take more time to develop, though 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, but not without some limitations.

Niklas Donges is an entrepreneur, technical writer and AI expert. He worked on an AI team of SAP for 1.5 years, after which he founded [Markov Solutions](#). The Berlin-based company specializes in artificial intelligence, machine learning and deep learning, offering customized AI-powered software solutions and consulting programs to various companies.

RELATED

[Read More About Data Science](#)

View remote jobs at top tech companies nationwide

BETA

NATIONALLY
LOCALLY
REMOTELY

FOR EMPLOYERS

How to Mind Goodhart's Law and Avoid Unintended Consequences

How to Write Nested List Comprehensions in Python

3 Reasons Data Scientists Need Linear Algebra

Data Science

Expert Contributors

View remote jobs at top tech companies nationwide

BETA

NATIONALLY
LOCALLY
REMOTELY

FOR EMPLOYERS



Expert Contributors

Built In's expert contributor network publishes thoughtful, solutions-oriented stories written by innovative tech professionals. It is the tech industry's definitive destination for sharing compelling, first-person accounts of problem-solving on the road to innovation.

[Learn More](#)

Great Companies Need Great People. **That's Where We Come In.**

[Recruit With Us](#)

Built In is the online community for startups and tech companies. Find startup jobs, tech news and events.

[About](#)[Our Story](#)

View remote jobs at top tech companies nationwide

BETA



FOR EMPLOYERS

Get Involved

[Recruit With Built In](#)

[Become an Expert Contributor](#)

Resources

[Customer Support](#)

[Share Feedback](#)

[Report a Bug](#)

[Remote Jobs in Atlanta](#)

[Remote Jobs in Dallas](#)

[Remote Jobs in DC](#)

[Browse Jobs](#)

Tech Hubs

[Built In Austin](#)

[Built In Boston](#)

[Built In Chicago](#)

[Built In Colorado](#)

[Built In LA](#)

[Built In NYC](#)

[View remote jobs at top tech companies nationwide](#)

BETA



FOR EMPLOYERS

© Built In 2021

[Accessibility Statement](#)

[Copyright Policy](#)

[Privacy Policy](#)

[Terms of Use](#)

[Do Not Sell My Personal Info](#)

[CA Notice of Collection](#)