**Machine Learning Algorithms Part3**

* + **1.1.1.4 Random Forest Regressor**

Code - <https://github.com/campusx-official/100-days-of-machine-learning/tree/main/day65-random-forest>

* **What is random forest regressor Algorithm**
* **Intuition behind the random forest regressor Algorithm**
* **How Random forest perform so well Bias and variance trade off:**
* **Bagging vs random forest**
* **Boosting vs Random forest**
* **Random forest Hyper parameters**
* **Random forest’s Hyperparameters tuning using grid search cv and RandomizedSearchCV**

**The Intuition behind Random Forest! Explained with example.**

If you have heard about the **Decision tree**, then you are not very far from understanding what random forests are. There are two keywords here - **random** and **forests**. Let us first understand what forest means. A **random forest** is a collection of many decision trees. Instead of relying on a single decision tree, you build many decision trees say 100 of them. And you know what a collection of trees is called - a forest. So you now understand why is it called a forest.

**Why is it called random then?**

Say our dataset has 1,000 rows and 30 columns. There are two levels of randomness in this algorithm:

* **At row level:** Each of these decision trees gets a random sample of the training data (say 10%) i.e. each of these trees will be trained independently on 100 randomly chosen rows out of 1,000 rows of data. Keep in mind that each of these decision trees is getting trained on 100 randomly chosen rows from the dataset i.e they are different from each other in terms of predictions.
* **At column level:** The second level of randomness is introduced at the column level. Not all the columns are passed into training each of the decision trees. Say we want only 10% of columns to be sent to each tree. This means a randomly selected 3 column will be sent to each tree. So for the first decision tree, maybe column C1, C2, and C4 were chosen. The next DT will have C4, C5, C10 as chosen columns, and so on.

Let us now understand how an interview selection process resembles a random forest algorithm. Each panel in the interview process is actually a decision tree. Each panel gives a result of whether the candidate is a pass or fail and then a majority of these results are declared as final. Say there were 5 panels, 3 said yes and 2 said no. The final verdict will be yes.

Something similar happens in the random forest as well. The results from each of the trees are taken and the final result is declared accordingly. **Voting** and **averaging** are used to predict in case of **classification** and **regression** respectively.

With the advent of huge computational power at our disposal, we hardly think for even a second before we apply random forests. And very conveniently our predictions are made. Let us try to understand other aspects of this algorithm.

***When is a random forest a poor choice relative to other algorithms?***

1. *Random forests don't train well on smaller datasets as it fails to pick on the pattern*. To simplify, say we know that 1 pen costs INR 1, 2 pens cost INR 2, 3 pens cost INR 6. In this case, linear regression will easily estimate the cost of 4 pens but random forests will fail to come up with a good estimate.
2. *There is a problem of interpretability with random forest*. You can't see or understand the relationship between the response and the independent variables. Understand that a random forest is a predictive tool and not a descriptive tool. You get variable importance but this may not suffice in many analyses of interests where the objective might be to see the relationship between response and the independent features.
3. The time taken to train random forests may sometimes be too huge as you train multiple decision trees. Also, in the case of a categorical variable, the time complexity increases exponentially. *For a categorical column with n levels, RF tries split at 2^n -1 points to find the maximal splitting point.*
4. In the case of a regression problem, *the range of values the response variable can take is determined by the values already available in the training dataset*. Unlike linear regression, decision trees and hence random forest can't take values outside the training data.

***What are the advantages of using random forest?***

* Since we are using multiple decision trees, the bias remains the same as that of a single decision tree. However, the variance decreases and thus we decrease the chances of overfitting.
* When all you care about is the predictions and want a quick and dirty way out, the random forest comes to the rescue. You don't have to worry much about the assumptions of the model or linearity in the dataset.
* **How Random forest perform so well Bias and variance trade off:**

The Bias–Variance Tradeoff Challenge  
High-variance, low-bias models (e.g., decision trees, KNN, SVM with complex kernels) can capture complex patterns but overfit easily. High-bias, low-variance models (e.g., linear or logistic regression) generalize stably but underfit when relationships are non‑linear.  
  
Traditionally, we had to choose between these extremes there was no single algorithm that balanced both low bias and low variance.  
  
How Random Forest Strikes the Balance:  
Random Forest combines many decision trees (each low-bias, high-variance) in two key ways:  
  
Bootstrap Aggregation (Bagging):  
Each tree is trained on a different random bootstrap sample of the data. This decorrelates the trees: noisy or outlier data points are unlikely to appear in the same way in every sample. When we average (or vote) across all trees, the individual overfitting “noise” tends to cancel out, reducing variance substantially.  
  
Random Feature Subsampling:  
At each split, a random subset of features is considered.  
This further decorrelates trees, since different trees see different features at different splits.  
It prevents any single strong predictor from dominating every tree, lowering variance without increasing bias much.  
  
Key Takeaways:  
Because each tree remains a deep, flexible model, bias stays low. But by averaging many weakly correlated trees, variance also drops giving Random Forest its reputation for robust, high‑accuracy performance without much tuning.

* **Bagging vs random forest**

The fundamental difference is that in Random forests, only a subset of features are selected at random out of the total and the best split feature from the subset is used to split each node in a tree, unlike in bagging where all features are considered for splitting a node.

**What exactly happens in Bagging?**

Bagging in general is an acronym-like work that is a portmanteau of Bootstrap and aggregation. In general, if you take a bunch of bootstrapped samples of your original dataset, fit models, and then average all the model predictions this is bootstrap aggregation i.e. Bagging.

This is done as a step within the Random forest model algorithm. The random forest creates bootstrap samples and across observations and for each fitted decision tree a random subsample of the covariates/features/columns are used in the fitting process.

The selection of each covariate is done with uniform probability in the original bootstrap paper. So if you had 100 covariates you would select a subset of these features each has a selection probability of 0.01. If you only had 1 covariate/feature you would select that feature with probability 1.

Overall RF provides more randomness over the Bagging and hence there are fewer chances of overfitting.

Please add if there are any other thoughts on this.

**Deeper Understanding**

**✅ What is Bagging?**

* “Bagging” = **Bootstrap Aggregating**.
* You build multiple models (usually decision trees) on **bootstrapped** (random with replacement) datasets.
* Final prediction = **average (regression)** or **majority vote (classification)**.

🔁 **Each tree sees a slightly different dataset**, but the same feature set is available for each split.

**✅ What is Random Forest?**

* Random Forest = **Bagging + Random Feature Subset Selection**
* At **each split in the tree**, only a **random subset of features** is considered — not all features.
* This reduces **correlation** among trees → makes the ensemble more diverse → reduces variance more effectively.
* **Random forest Hyper parameters**

<https://www.kaggle.com/code/nargisbegum82/hyperparameter-tuning-in-random-forests>

Hyperparameters are configurations that cannot be learnt from the regular data that we provide to the algorithm, these are inbuilt to the algorithm and each algorithm has its own predefined set of hyperparameters.  Hyperparameters are often tuned for increasing model accuracy, and we can use various methods such as GridSearchCV, RandomizedSearchCV as explained in the article <https://www.geeksforgeeks.org/hyperparameter-tuning/>.

A deep understanding of hyperparameters is required because they are responsible for deciding how quickly a model can fit onto the data to produce accurate results. On the other hand, not finding the optimal values of hyperparameters can also result in less accuracy because of overfitting issue. Therefore, we will be having a closer look at the hyperparameters of random forest classifier to have a better understanding of the inbuilt hyperparameters:

* **n\_estimators:**We know that a random forest is nothing but a group of many decision trees, the n\_estimator parameter controls the number of trees inside the classifier. We may think that using many trees to fit a model will help us to get a more generalized result, but this is not always the case. However, it will not cause any overfitting but can certainly increase the time complexity of the model. **The default number of estimators is 100**in scikit-learn.
* **max\_depth:**It governs the maximum height upto which the trees inside the forest can grow. It is one of the most important hyperparameters when it comes to increasing the accuracy of the model, as we increase the depth of the tree the model accuracy increases upto a certain limit but then it will start to decrease gradually because of overfitting in the model. It is important to set its value appropriately to avoid overfitting. **The default value is set to** **None,**None specifies that the nodes inside the tree will continue to grow until all leaves become pure or all leaves contain less than **min\_samples\_split** (another hyperparameter).
* **min\_samples\_split:**It specifies the minimum amount of samples an internal node must hold in order to split into further nodes. If we have a very low value of min\_samples\_splits then, in this case, our tree will continue to grow and start overfitting. By increasing the value of min\_samples\_splits we can decrease the total number of splits thus limiting the number of parameters in the model and thus can aid in reducing the overfitting in the model. However, the value should not be kept very large that a number of parameters drop extremely causing the model to underfit. We generally keep min\_samples\_split value between 2 and 6. **However, the default value is set to 2.**
* **min\_samples\_leaf:** It specifies the minimum amount of samples that a node must hold after getting split. It also helps to reduce overfitting when we have ample amount of parameters. Less number of parameters can lead to overfitting also, we should keep in mind that increasing the value to a large number can lead to less number of parameters and in this case model can underfit also. **The default value is set to 1.**
* **max\_features:**Random forest takes random subsets of features and tries to find the best split.  max\_features helps to find the number of features to take into account in order to make the best split. It can take four values "**auto**", "**sqrt**", "**log2**" and **None**.
  + In case of auto: considers max\_features = sqrt(n\_features)
  + In case of sqrt: considers max\_features = sqrt(n\_features), it is same as auto
  + In case of log2: considers max\_features = log2(n\_features)
  + In case of None: considers max\_features = n\_features
* **max\_leaf\_nodes:**It sets a limit on the splitting of the node and thus helps to reduce the depth of the tree, and effectively helps in reducing overfitting. If the value is set to None, the tree continues to grow infinitely.
* **max\_samples:**This hyperparameter helps to choose maximum number of samples from the training dataset to train each individual tree.

These are the major hyperparameters that are present implicitly in the random forest classifier which is required to be tuned in order to increase the accuracy of our training model.

* **Random forest’s Hyperparameters tuning using grid search cv and RandomizedSearchCV**

<https://www.geeksforgeeks.org/random-forest-hyperparameter-tuning-in-python/>

[Random Forest](https://www.geeksforgeeks.org/random-forest-algorithm-in-machine-learning/) is one of the most popular machine learning algorithms used for both classification and regression tasks. It works by building multiple decision trees and combining their outputs to improve accuracy and control overfitting. While Random Forest is a robust model, fine-tuning its hyperparameters such as the number of trees, maximum depth and feature selection can improve its prediction and performance and in this article we will learn how we can do it.

Since we are talking about Random Forest Hyperparameters let us see what different Hyperparameters can be Tuned:

**1. n\_estimators:**It defines the number of trees in the forest. More trees typically improve model performance but increase computational cost. In the below example it takes 100 trees.

***By default:*** *n\_estimators=100*

**2. max\_features:** Limits the number of features to consider when splitting a node. This helps control overfitting.

***By default:*** *max\_features="sqrt" [available: ["sqrt", "log2", None}]*

* **sqrt**: Selects the square root of the total features. This is a common setting to reduce overfitting and speed up the model.
* **log2:** This option selects the base-2 logarithm of the total number of features. It provide more randomness and reduce overfitting more than the square root option.
* **None:** If None is chosen the model uses all available features for splitting each node. This increases the model's complexity and may cause overfitting, especially with many features.

**3. max\_depth**: Controls the maximum depth of each tree. A shallow tree may underfit while a deep tree may overfit. So choosing right value of it is important.

***By default:*** *max\_depth=None*

**4. max\_leaf\_nodes:** Limits the number of leaf nodes in the tree hence controlling its size and complexity. None means it takes an unlimited number of nodes.

***By default:*** *max\_leaf\_nodes = None*

**5. max\_sample**: Apart from the features, we have a large set of training datasets. max\_sample determines how much of the dataset is given to each individual tree. None means data.shape[0] is taken.

***By default:*** *max\_sample = None*

**6. min\_sample\_split:** Specifies the minimum number of samples required to split an internal node. In the below example every node has 2 subnodes.

***By default:*** *min\_sample\_split = 2*

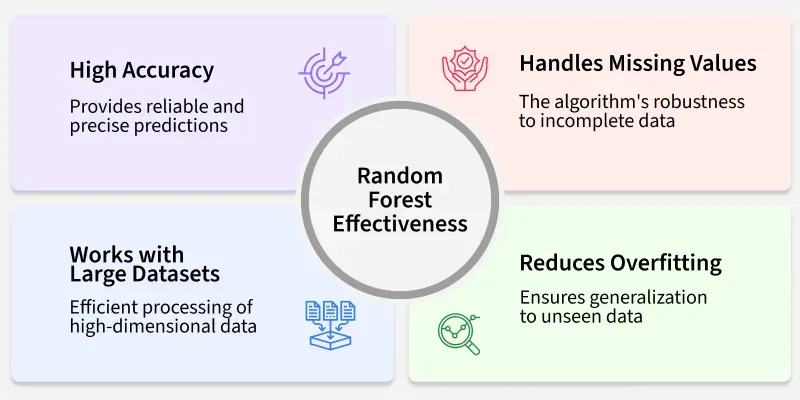
**4) Random Forest Regressor**

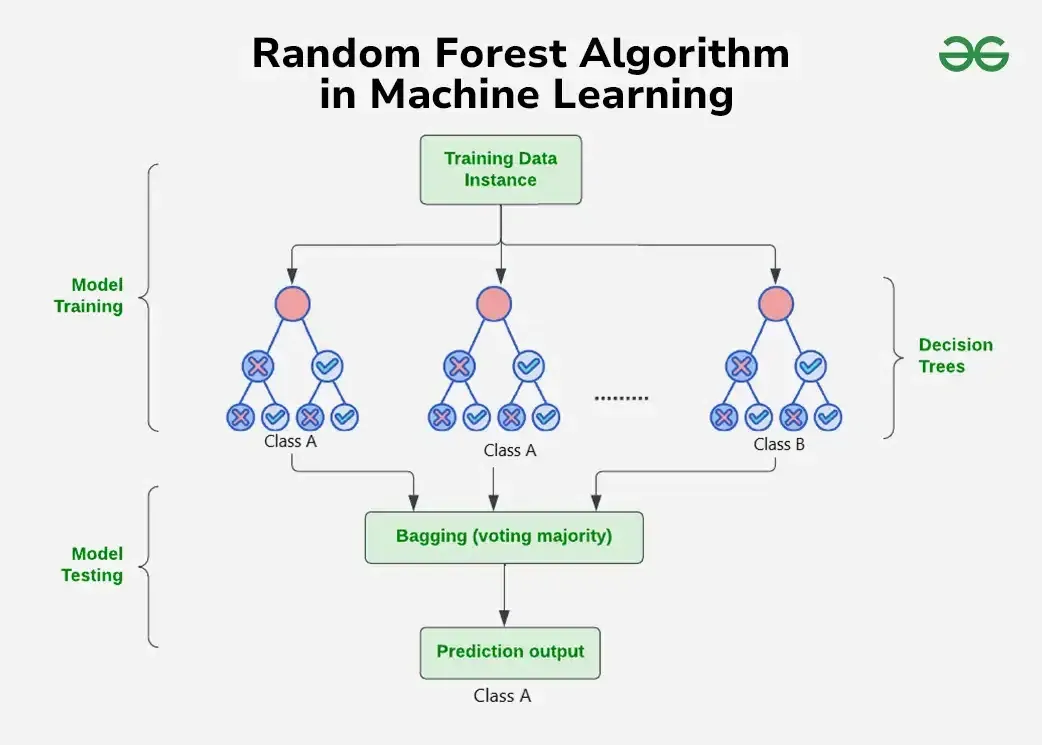
**📘 What is it?**

A **Random Forest Regressor** is an **ensemble learning method** that builds multiple decision trees and averages their outputs to improve predictive accuracy and reduce overfitting.

It's part of the **Bagging** family (Bootstrap Aggregation).

Random Forest is a machine learning algorithm that uses many decision trees to make better predictions. Each tree looks at different random parts of the data and their results are combined by voting for classification or averaging for regression. This helps in improving accuracy and reducing errors.





**Working of Random Forest Algorithm**

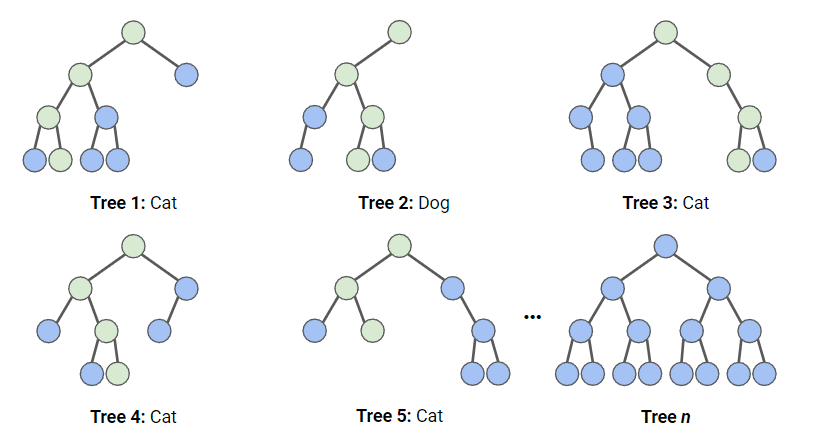
**How Random Forest Classification Works**

Imagine you have a complex problem to solve, and you gather a group of experts from different fields to provide their input. Each expert provides their opinion based on their expertise and experience. Then, the experts would vote to arrive at a final decision.

In a random forest classification, multiple decision trees are created using different random subsets of the data and features. Each decision tree is like an expert, providing its opinion on how to classify the data. Predictions are made by calculating the prediction for each decision tree and then taking the most popular result. (For regression, predictions use an averaging technique instead.)

In the diagram below, we have a random forest with n decision trees, and we’ve shown the first 5, along with their predictions (either “Dog” or “Cat”). Each tree is exposed to a different number of features and a different sample of the original dataset, and as such, every tree can be different. Each tree makes a prediction.

Looking at the first 5 trees, we can see that 4/5 predicted the sample was a Cat. The green circles indicate a hypothetical path the tree took to reach its decision. The random forest would count the number of predictions from decision trees for Cat and for Dog, and choose the most popular prediction.

*Illustration of how random forest classification works. Image by Author*

* **Create Many Decision Trees:** The algorithm makes many [decision trees](https://www.geeksforgeeks.org/decision-tree/) each using a random part of the data. So every tree is a bit different.
* **Pick Random Features:** When building each tree it doesn’t look at all the features (columns) at once. It picks a few at random to decide how to split the data. This helps the trees stay different from each other.
* **Each Tree Makes a Prediction:** Every tree gives its own answer or prediction based on what it learned from its part of the data.
* **Combine the Predictions:**
  + For **classification** we choose a category as the final answer is the one that most trees agree on i.e majority voting.
  + For **regression** we predict a number as the final answer is the average of all the trees predictions.
* **Why It Works Well:** Using random data and features for each tree helps avoid overfitting and makes the overall prediction more accurate and trustworthy.

*Random forest is also a ensemble learning technique which you can learn more about from:* [*Ensemble Learning*](https://www.geeksforgeeks.org/a-comprehensive-guide-to-ensemble-learning/)

**Key Features of Random Forest**

* **Handles Missing Data:** It can work even if some data is missing so you don’t always need to fill in the gaps yourself.
* **Shows Feature Importance:** It tells you which features (columns) are most useful for making predictions which helps you understand your data better.
* **Works Well with Big and Complex Data:** It can handle large datasets with many features without slowing down or losing accuracy.
* **Used for Different Tasks:** You can use it for both **classification** like predicting types or labels and **regression** like predicting numbers or amounts.

**Assumptions of Random Forest**

* **Each tree makes its own decisions**: Every tree in the forest makes its own predictions without relying on others.
* **Random parts of the data are used**: Each tree is built using random samples and features to reduce mistakes.
* **Enough data is needed**: Sufficient data ensures the trees are different and learn unique patterns and variety.
* **Different predictions improve accuracy**: Combining the predictions from different trees leads to a more accurate final result.

**Advantages of Random Forest Regression**

* **Handles Non-Linearity**: It can capture complex, non-linear relationships in the data that other models might miss.
* **Reduces Overfitting**: By combining multiple decision trees and averaging predictions it reduces the risk of overfitting compared to a single decision tree.
* **Robust to Outliers**: Random Forest is less sensitive to outliers as it aggregates the predictions from multiple trees.
* **Works Well with Large Datasets**: It can efficiently handle large datasets and high-dimensional data without a significant loss in performance.
* **Handles Missing Data**: Random Forest can handle missing values by using surrogate splits and maintaining high accuracy even with incomplete data.
* **No Need for Feature Scaling**: Unlike many other algorithms Random Forest does not require normalization or scaling of the data.

**Disadvantages of Random Forest Regression**

* **Complexity**: It can be computationally expensive and slow to train especially with a large number of trees and high-dimensional data. Due to this it may not be suitable for real-time predictions especially with a large number of trees.
* **Less Interpretability**: Since it uses many trees it can be harder to interpret compared to simpler models like linear regression or decision trees.
* **Memory Intensive**: Storing multiple decision trees for large datasets require significant memory resources.
* **Overfitting on Noisy Data**: While Random Forest reduces overfitting, it can still overfit if the data is highly noisy especially with a large number of trees.
* **Sensitive to Imbalanced Data**: It may perform poorly if the dataset is highly imbalanced like one class is significantly more frequent than another.

**Random Forest in Classification and Regression**

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](https://builtin.com/data-science/random-forest-python) 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](https://builtin.com/software-engineering-perspectives/tree-traversal), 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 a random forest classifier, 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).

**Random Forest Models vs. Decision Trees**

While a random forest model is a collection of decision trees, there are [some differences](https://builtin.com/data-science/supervised-machine-learning-classification).

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, 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 their 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](https://builtin.com/articles/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.

**A Real-Life Example of Random Forest**

Andrew wants to decide where to go during his one-year vacation, so he asks the people who know him best for suggestions. The first friend he seeks out asks him about the likes and dislikes of his past travels. Based on the answers, he will give Andrew some advice.

This is a typical [decision tree algorithm approach](https://builtin.com/data-science/classification-tree). Andrew’s friend created rules to guide his decision about what he should recommend, by using Andrew’s answers.

Afterward, 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 his friends recommend the most to him, which is the typical random forest algorithm approach.

Building, using and evaluating random forests. | Video: StatQuest with Josh Starmer

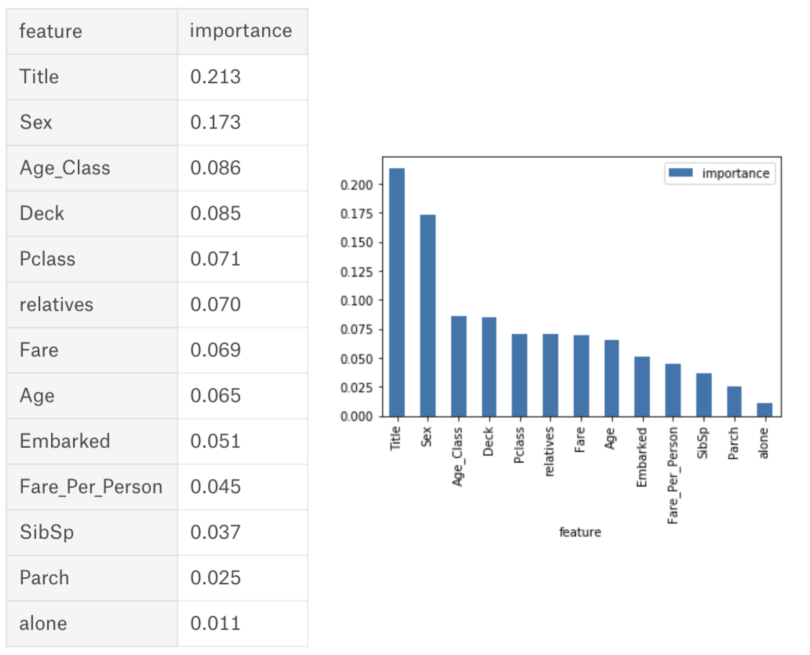
**Random Forest Feature Importance**

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](https://builtin.com/machine-learning/scikit-learn-guide" \t "_blank) provides a great tool for this that measures a [feature’s importance](https://builtin.com/data-science/feature-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.

If you don’t know how a decision tree works or 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.”

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](https://builtin.com/data-science/model-fit) 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](https://www.kaggle.com/niklasdonges/end-to-end-project-with-python).



# Data Processing

import pandas as pd

import numpy as np

# Modelling

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, precision\_score, recall\_score, ConfusionMatrixDisplay

from sklearn.model\_selection import RandomizedSearchCV, train\_test\_split

from scipy.stats import randint

# Tree Visualisation

from sklearn.tree import export\_graphviz

from IPython.display import Image

import graphviz

**The Random Forest Workflow**

To fit and train this model, we’ll be following [**The Machine Learning Workflow**](https://www.datacamp.com/blog/a-beginner-s-guide-to-the-machine-learning-workflow) infographic; however, as our data is pretty clean, we won’t be carrying out every step. We will do the following:

* Feature engineering
* Split the data
* Train the model
* Hyperparameter tuning
* Assess model performance

**Preprocessing Data for our Random Forest Classifier**

Tree-based models are much more robust to outliers than linear models, and they do not need variables to be normalized to work. As such, we need to do very little preprocessing on our data.

* We will map our default column, which contains no and yes, to 0s and 1s, respectively. We will treat unknown values as no for this example.
* We will also map our target, y, to 1s and 0s.

bank\_data['default'] = bank\_data['default'].map({'no':0,'yes':1,'unknown':0})

bank\_data['y'] = bank\_data['y'].map({'no':0,'yes':1})

[**Powered By**](https://www.datacamp.com/datalab)

**Splitting the Data**

When training any supervised learning model, it is important to split the data into training and test data. The training data is used to fit the model. The algorithm uses the training data to learn the relationship between the features and the target. The test data is used to evaluate the performance of the model.

The code below splits the data into separate variables for the features and target, then splits into training and test data.

# Split the data into features (X) and target (y)

X = bank\_data.drop('y', axis=1)

y = bank\_data['y']

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

[**Powered By**](https://www.datacamp.com/datalab)

**Fitting and Evaluating the Random Forest Model**

We first create an instance of the Random forest model with the default parameters. We then fit this to our training data. We pass both the features and the target variable so the model can learn.

rf = RandomForestClassifier()

rf.fit(X\_train, y\_train)

[**Powered By**](https://www.datacamp.com/datalab)

At this point, we have a trained random forest model, but we need to find out whether it makes accurate predictions.

y\_pred = rf.predict(X\_test)

[**Powered By**](https://www.datacamp.com/datalab)

The simplest way to evaluate this model is using accuracy; we check the predictions against the actual values in the test set and count up how many the model got right.

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

[**Powered By**](https://www.datacamp.com/datalab)

Output:

Accuracy: 0.888

[**Powered By**](https://www.datacamp.com/datalab)

This is a pretty good score! However, we may be able to do better by optimizing our hyperparameters.

**Visualizing the Results**

We can use the following code to visualize our first 3 trees.

# Export the first three decision trees from the forest

for i in range(3):

tree = rf.estimators\_[i]

dot\_data = export\_graphviz(tree,

feature\_names=X\_train.columns,

filled=True,

max\_depth=2,

impurity=False,

proportion=True)

graph = graphviz.Source(dot\_data)

display(graph)

**Random Forest interview question and answers**

**Basic Questions**

**1. What is a Random Forest?**

**Answer:**  
Random Forest is an ensemble learning technique that builds multiple decision trees and merges their results (via majority voting for classification or averaging for regression) to improve accuracy and reduce overfitting.

**2. How does Random Forest differ from a decision tree?**

**Answer:**  
While a decision tree builds one tree based on the full dataset, Random Forest builds multiple trees using:

* Random subsets of data (bagging)
* Random subsets of features at each split (feature sampling)

This randomness leads to better generalization and lower overfitting.

**3. What is bagging in Random Forest?**

**Answer:**  
Bagging (Bootstrap Aggregating) involves training each decision tree on a random subset of the training data (with replacement). This ensures diversity among trees and reduces variance.

**4. What are the key hyperparameters of Random Forest?**

**Answer:**

* n\_estimators: Number of trees
* max\_depth: Maximum depth of each tree
* min\_samples\_split: Minimum samples required to split a node
* min\_samples\_leaf: Minimum samples at a leaf node
* max\_features: Number of features to consider when looking for the best split

**5. What are the advantages of Random Forest?**

**Answer:**

* High accuracy
* Reduces overfitting
* Handles missing values and categorical features well
* Works well for both classification and regression tasks

**🔹 Intermediate Questions**

**6. How does Random Forest handle missing values?**

**Answer:**  
Random Forest can handle missing values by:

* Ignoring them during split decisions
* Using surrogate splits
* Imputing them internally during training

**7. How do you measure feature importance in Random Forest?**

**Answer:**  
Random Forest provides **feature importance** using:

* **Gini Importance** (Mean Decrease in Impurity)
* **Permutation Importance** (based on the decrease in model performance when a feature is shuffled)

**8. Can Random Forest be used for time-series data?**

**Answer:**  
Not directly. Random Forest doesn’t account for temporal dependencies, but you can:

* Create lag features manually
* Use it as part of a pipeline with feature engineering

**9. What is OOB (Out-of-Bag) error in Random Forest?**

**Answer:**  
Out-of-Bag error is a validation technique where each tree is evaluated using the data not included in its bootstrap sample. It's like cross-validation and provides an unbiased estimate of model accuracy.

**10. How do you prevent overfitting in Random Forest?**

**Answer:**

* Limit tree depth (max\_depth)
* Reduce number of features (max\_features)
* Use fewer trees (n\_estimators)
* Use min\_samples\_split and min\_samples\_leaf constraints

**🔹 Advanced/Tricky Questions**

**11. Why does Random Forest perform better than a single decision tree?**

**Answer:**  
Random Forest reduces **variance** by averaging multiple trees, each trained on different data and feature subsets. This results in better generalization and robustness to noise.

**12. Can you explain how Random Forest introduces randomness?**

**Answer:**

* **Data randomness**: Each tree is trained on a different bootstrapped dataset.
* **Feature randomness**: A random subset of features is considered at each split.

This combination increases diversity among trees.

**13. What if you increase the number of trees indefinitely?**

**Answer:**

* Model variance continues to reduce.
* Eventually, performance plateaus.
* After a certain point, adding more trees won’t help much and just increases computation.

**14. What happens if you set max\_features = 1 in a Random Forest classifier?**

**Answer:**  
At each split, only one feature is considered, which increases tree diversity and may help reduce overfitting — but can also hurt accuracy if features are informative.

**15. How does Random Forest handle imbalanced datasets?**

**Answer:**

* Class weighting (class\_weight=’balanced’)
* Sampling techniques (SMOTE, undersampling)
* Using evaluation metrics like F1-score, AUC instead of accuracy