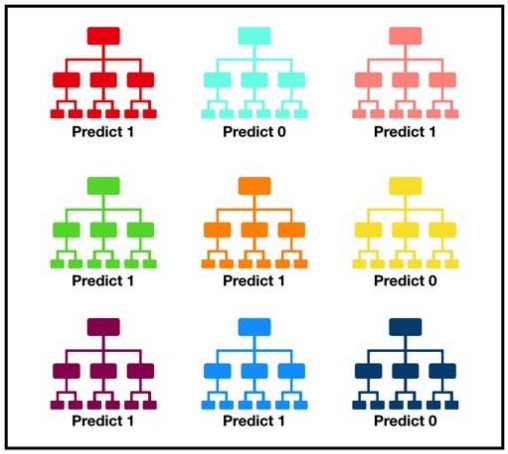
1.8. Random Forest

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction (see figure below).



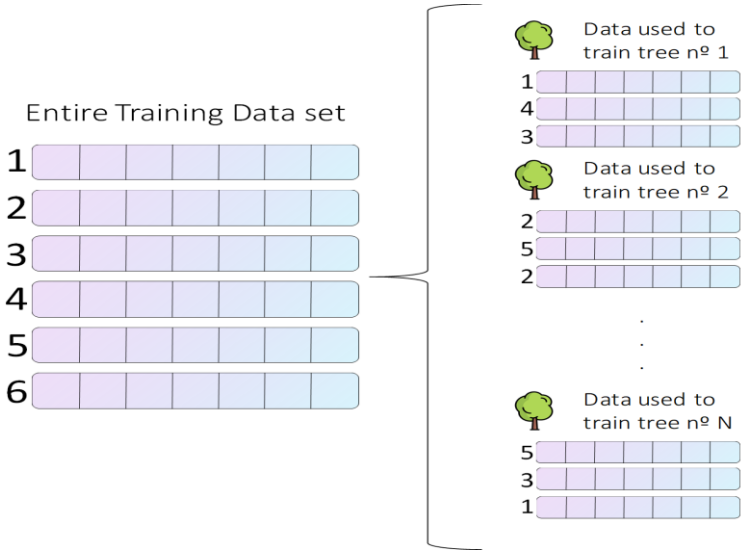
Tally: Six 1s and Three 0s
Prediction: 1

A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models. The reason for this wonderful effect is that, while some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction.

Okay, we got that, a Random Forest is a collection of individual trees. But why the name Random? Where is the Randomness? Lets find out by learning how a Random Forest model is built.

To build a Random Forest we have to train N decision trees. Do we train the trees using the same data all the time? Do we use the whole data set? **Nope**.

To train each individual tree, we pick a random sample of the entire Data set, like shown in the following figure.



From looking at this figure, various things can be deduced. First of all, the size of the data used to train each individual tree does not have to be the size of the whole data set. Also, a data point can be present more than once in the data used to train a single tree (like in tree nº two).

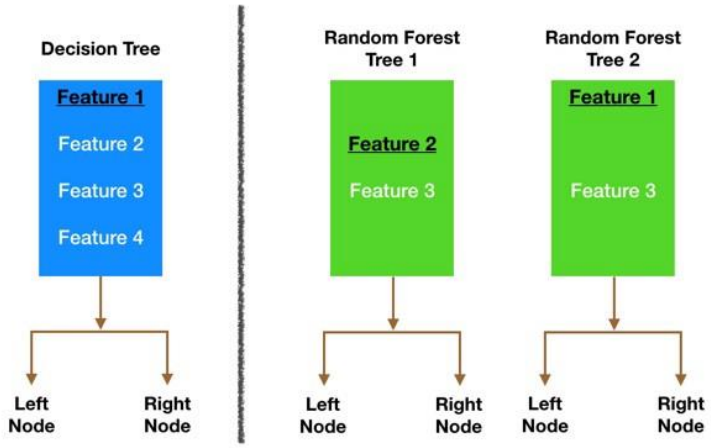
This is called Sampling with Replacement or ***Bootstrapping***: each data point is picked randomly from the whole data set, and a data point can be picked more than once.

NOTE:

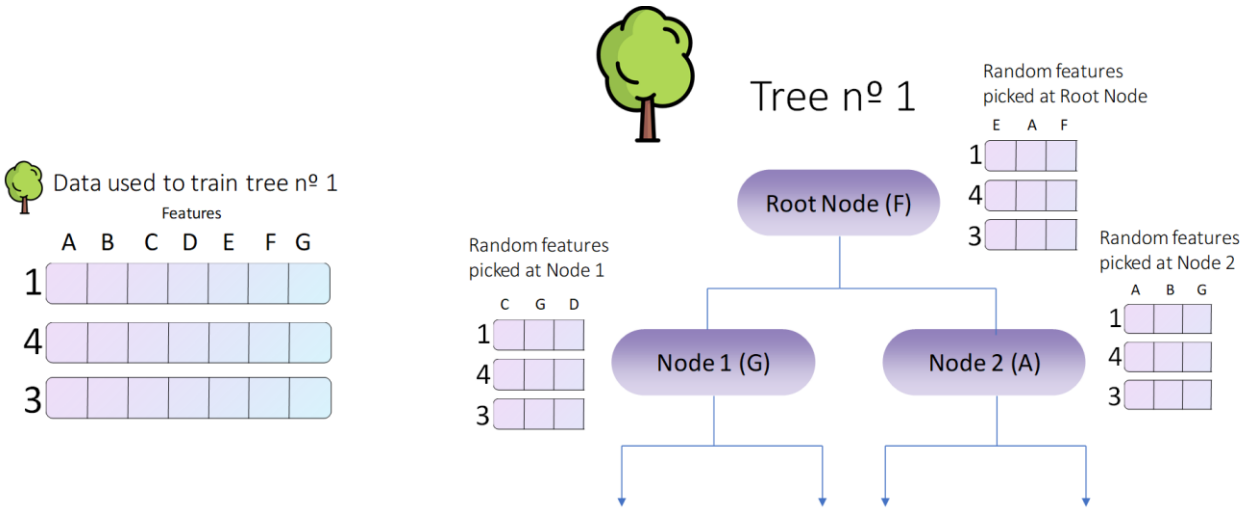
In practice, by default most Random Forest implementations (like the one from Scikit-Learn) pick the sample of the training data used for each tree to be the same size as the original data set (however it is not the same data set, remember that we are picking random samples).

Feature Randomness

In a normal decision tree, when it is time to split a node, we consider every possible feature and pick the one that produces the most separation between the observations in the left node vs. those in the right node. In contrast, each tree in a random forest can pick only from a random subset of features. This forces even more variation amongst the trees in the model.



Feature randomness can also be achieved by randomly selecting certain features to evaluate at each node.



As you can see from the previous image, at each node we evaluate only a subset of all the initial features. For the root node we take into account E, A and F (and F wins). In Node 1 we consider C, G and D (and G wins). Lastly, in Node 2 we consider only A, B, and G (and A wins). We would carry on doing this until we built the whole tree.

So in our random forest, we end up with trees that are not only trained on different sets of data but also use different features to make decisions. ***The greater the tree diversity, the better: we reduce the variance, and get a better performing model.***

There is a direct relationship between the number of trees in the forest and the results it can get: **the larger the number of trees, the more accurate the result**.

We have now learned how to build a single decision tree with slight variations. Now, we would repeat this for the N trees.

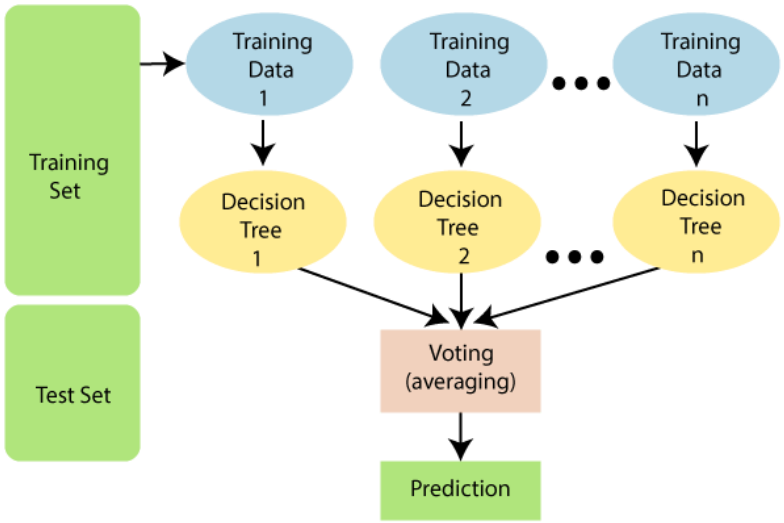
In conclusion, the whole process goes as follows:

1. Create a bootstrapped data set for each tree.

2. Create a decision tree using its corresponding data set, but at each node use a random sub sample of variables or features to split on.

3. Repeat all these three steps hundreds of times to build a massive forest with a wide variety of trees. This variety is what makes a Random Forest way better than a single decision tree.

Making predictions with a Random Forest is very easy. We just have to take each of our individual trees, pass the observation for which we want to make a prediction through them, get a prediction from every tree (summing up to N predictions) and then obtain an overall, aggregated prediction.



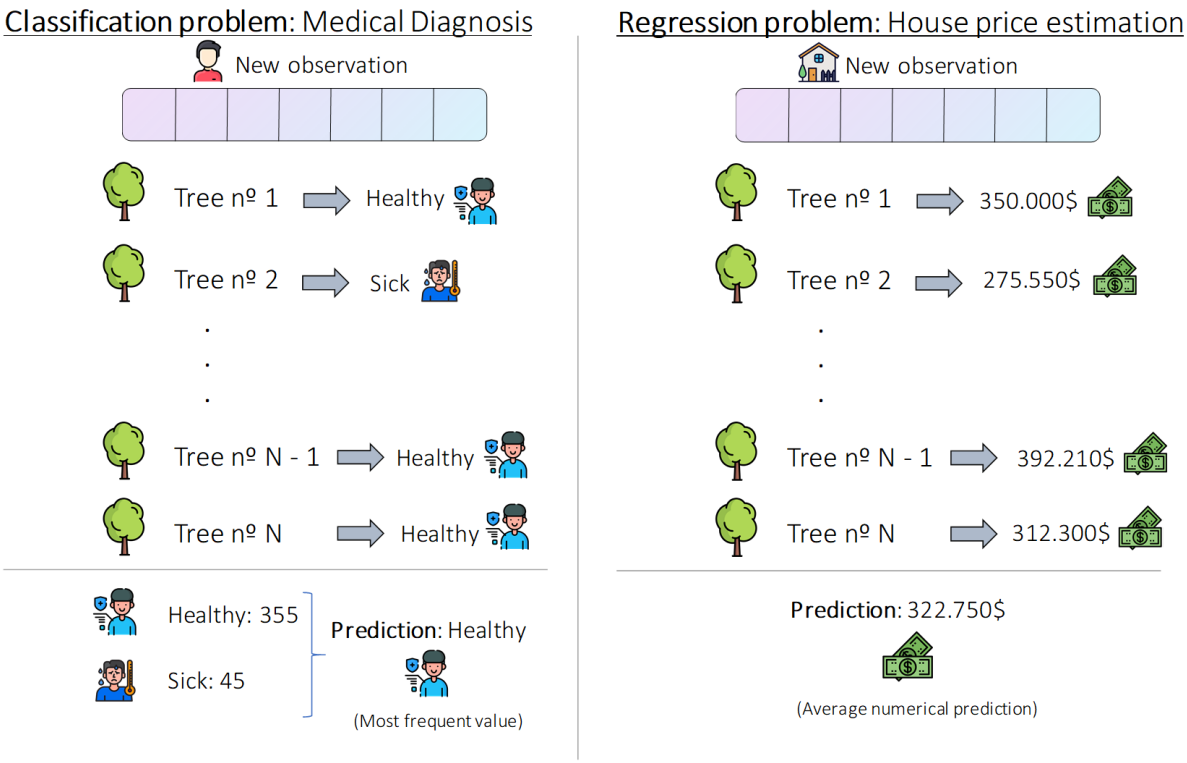
Bootstrapping the data and then using an aggregate to make a prediction is called Bagging.

The random forest prediction steps are as below:

1. Takes the test features and use the rules of each randomly created decision tree to predict the outcome and stores the predicted outcome (target)

2. Calculate the votes for each predicted target

3. Consider the high voted predicted target as the final prediction from the random forest algorithm



The previous image illustrates this very simple procedure. For the classification problem we want to predict if a certain patient is sick or healthy. For this we pass his medical record and other information through each tree of the random forest, and obtain N predictions (400 in our case). In our example 355 of the trees say that the patient is healthy and 45 say that the patient is sick, therefore the forest decides that the patient is healthy.

For the regression problem we want to predict the price of a certain house. We pass the characteristics of this new house through our N trees, getting a numerical prediction from each of them. Then, we calculate the average of these predictions and get the final value of 322.750$.

Although Random Forest models don't offer as much interpret ability as a single tree, their performance is a lot better, and we don't have to worry so much about perfectly tuning the parameters of the forest as we do with individual trees.

The difference between Random Forest algorithm and the decision tree algorithm is that in Random Forest, the processes of finding the root node and splitting the feature nodes will run randomly.

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 sklearns built-in random forest function.

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.

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

Feature Importance

Feature importance refers to techniques that assign a score to input features based on how useful they are at predicting a target variable.

The scores are useful and can be used in a range of situations in a predictive modeling problem, such as:

- ✓ Better understanding the data.
- ✓ Better understanding a model.
- ✓ Reducing the number of input features.

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.

We can use the Random Forest algorithm for feature importance implemented in scikit-learn as the `RandomForestRegressor` and `RandomForestClassifier` classes.

After being fit, the model provides a `feature_importances_` property that can be accessed to retrieve the relative importance scores for each input feature.

Step 1: First, we must train our Random Forest model

Step 2: Once the Random Forest model is built, we can directly extract the feature importance with the forest of trees using the `feature_importances_` attribute of the `RandomForestClassifier` model, like so:

Step 3: Step 2 will return an array full of numbers, and nothing we can easily interpret. To build a Random Forest feature importance plot, and easily see the Random Forest importance score reflected in a table, we have to create a Data Frame out of it.

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.

Feature importance scores play an important role in a predictive modeling project, including providing insight into the data, insight into the model, and the basis for dimensionality reduction and feature selection that can improve the efficiency and effectiveness of a predictive model on the problem.

CAUTION:

Using Random Forest to calculate feature importance tends to inflate the relevance of continuous features or high cardinality categorial variables versus those discrete variables with fewer available values.

This is because these kinds of variables, because of their nature have a higher chance of appearing more than once in an individual tree, which contributes to an increase in their importance. If a binary feature is really relevant though, it will still be reflected in the feature importance ranking [1].