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

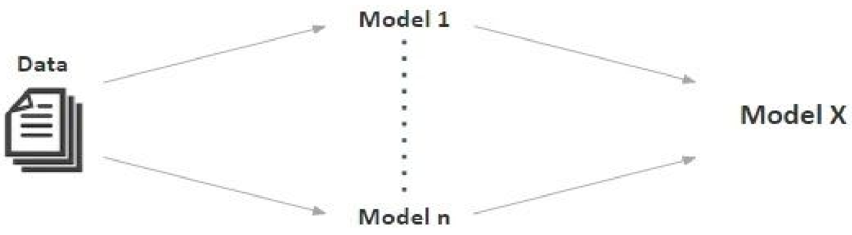
A random forest algorithm combines multiple decision trees to generate the final results. This process of combining more than one model to take the final decision is termed as ensemble learning.

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

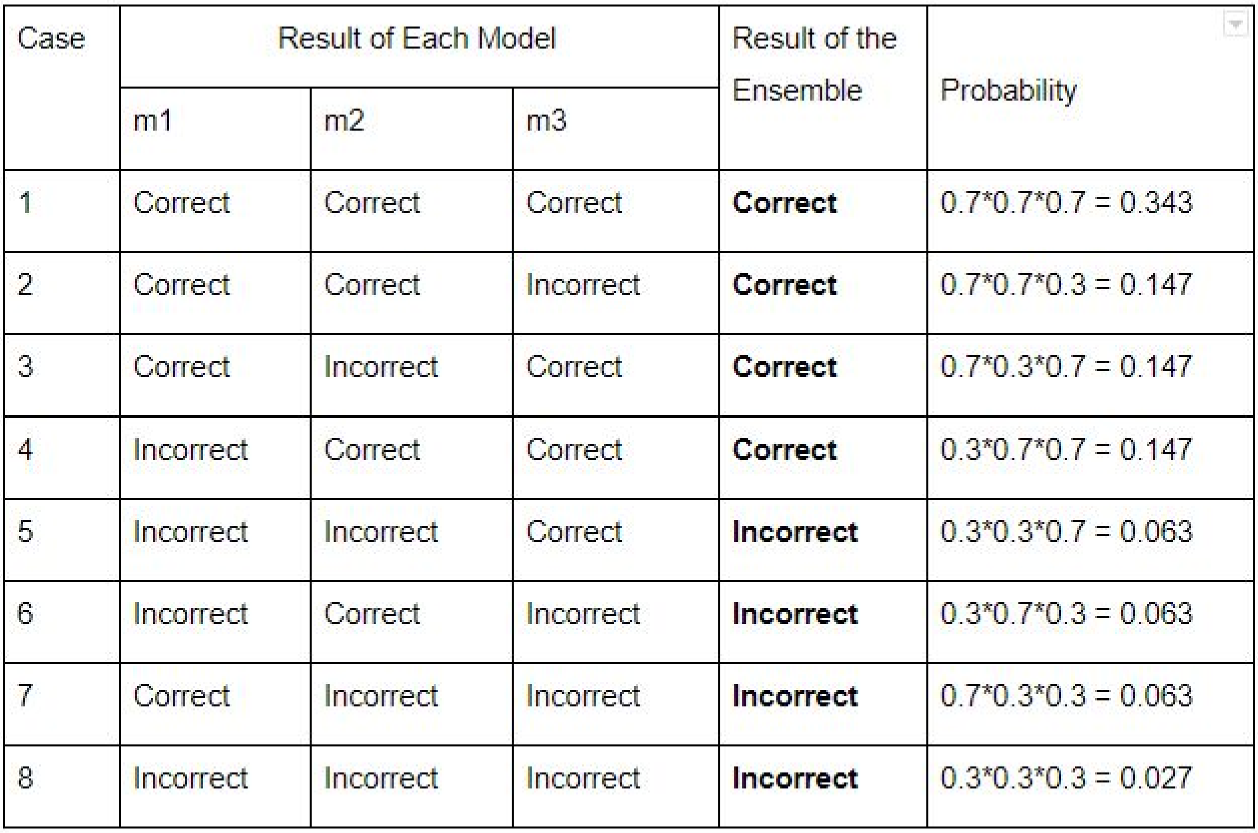
- Use different features sets for each of the models

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 x 0.7 x 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 x 0.7 x 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*0.7*0.7 = 0.343
2	Correct	Correct	Incorrect	**Correct**	0.7*0.7*0.3 = 0.147
3	Correct	Incorrect	Correct	**Correct**	0.7*0.3*0.7 = 0.147
4	Incorrect	Correct	Correct	**Correct**	0.3*0.7*0.7 = 0.147
5	Incorrect	Incorrect	Correct	**Incorrect**	0.3*0.3*0.7 = 0.063
6	Incorrect	Correct	Incorrect	**Incorrect**	0.3*0.7*0.3 = 0.063
7	Correct	Incorrect	Incorrect	**Incorrect**	0.7*0.3*0.3 = 0.063
8	Incorrect	Incorrect	Incorrect	**Incorrect**	0.3*0.3*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

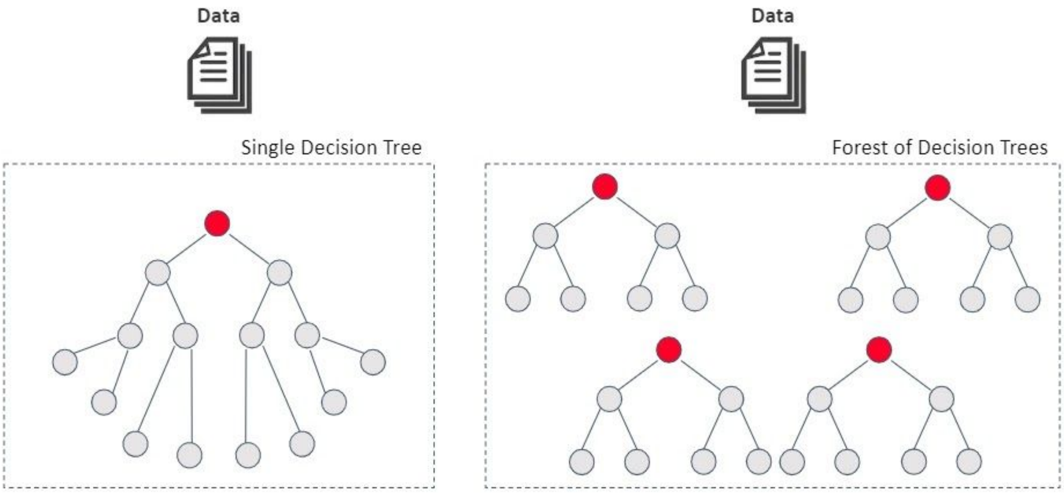
The random forest algorithm is an ensemble of decision trees that uses bagging to generate different base models. So far, random forest has been the most successful model among the bagging ensembles. They are essentially ensembles of a number of decision trees. You can create a large number of models (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.

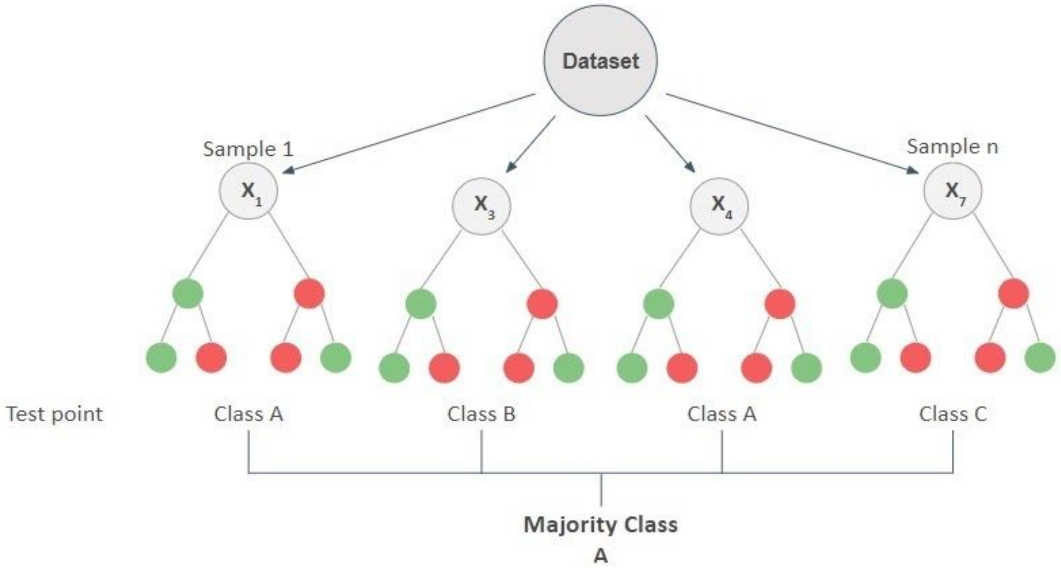
In case of a classification problem it takes a majority vote on the predictions made by models for every data point and provides the output as the class that has received the highest number of votes. In the case of regression, it will be the average of all the predicted values available for a particular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

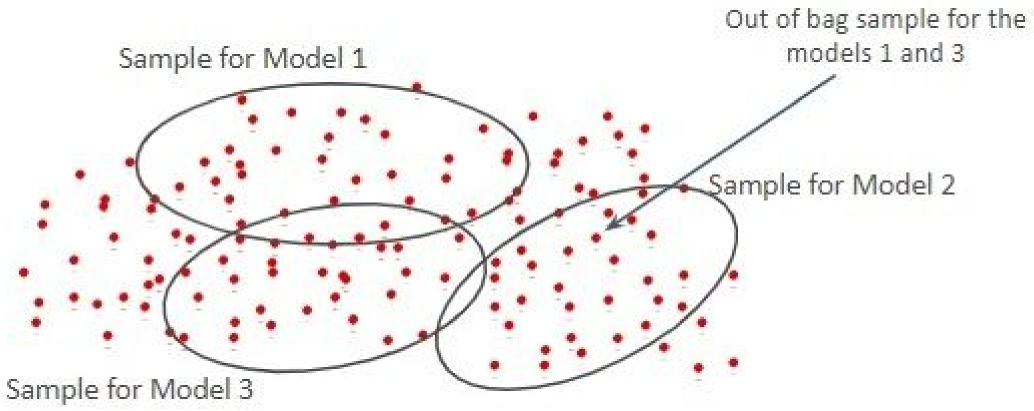


Random Forest Algorithm

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.

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

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 depends 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.

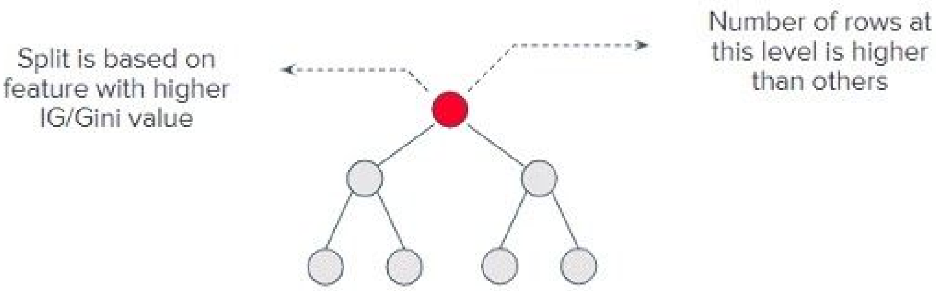
The disadvantages of the random forest model are as follows:

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.

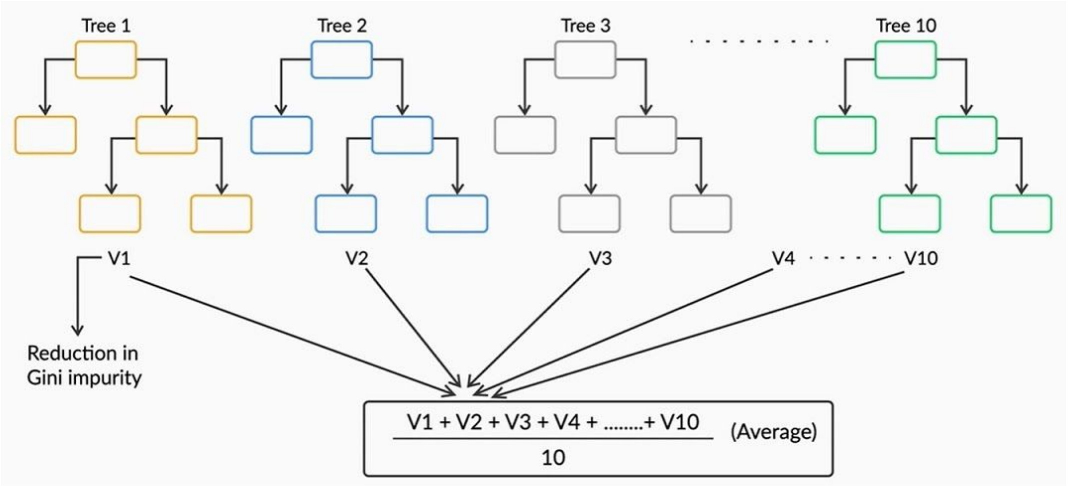
Feature Importance in Random Forests

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.


Split is based on feature with higher IG/Gini value
Number of rows at this level is higher than others

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.


Tree 1
Tree 2
Tree 3
Tree 10
V1
V2
V3
V4
V10
Reduction in Gini impurity
V1 + V2 + V3 + V4 + ...+ V10 / 10 (Average)

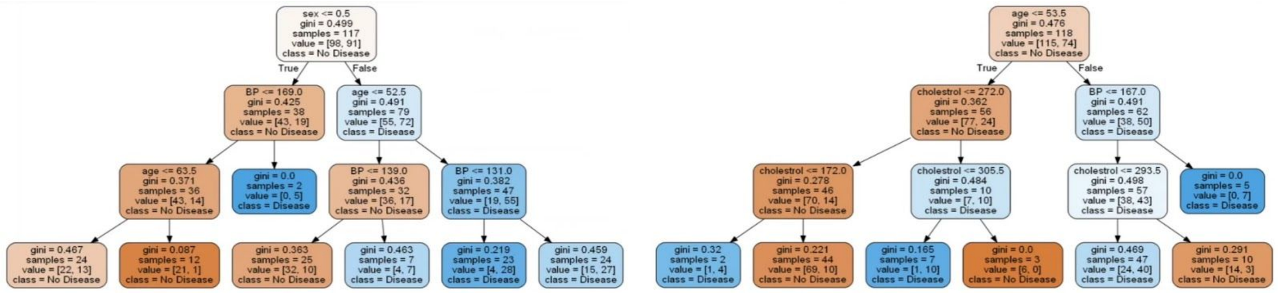
For each variable, the sum of the information gain across every tree of the forest is accumulated every time a variable is chosen to split a node. The value is then divided by the number of base trees in which the feature is involved to give the final average value.

Python: Heart Disease Prediction

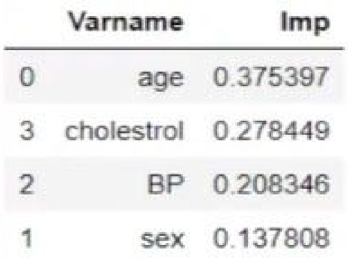
In this segment, you learnt how to implement random forests in Python using the sklearn library. You worked with the heart disease data to build a random forest model to predict whether a person has heart disease or not. The data lists the results of various tests that were conducted on patients along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


sex <= 0.5 gini = 0.499 samples = 117 value = [98, 91] class = No Disease
True
False
BP <= 169.0 gini = 0.425 samples = 38 value = [43, 19] class = No Disease
age <= 52.5 gini = 0.491 samples = 79 value = [55, 72] class = Disease
age <= 63.5 gini = 0.371 samples = 36 value = [43, 14] class = No Disease
gini = 0.0 samples = 2 value = [0, 5] class = Disease
BP <= 139.0 gini = 0.436 samples = 32 value = [36, 17] class = No Disease
BP <= 131.0 gini = 0.382 samples = 47 value = [19, 55] class = Disease
gini = 0.467 samples = 24 value = [22, 13] class = No Disease
gini = 0.087 samples = 12 value = [21, 1] class = No Disease
gini = 0.363 samples = 25 value = [32, 10] class = No Disease
gini = 0.463 samples = 7 value = [4, 7] class = Disease
gini = 0.219 samples = 23 value = [4, 28] class = Disease
gini = 0.459 samples = 24 value = [15, 27] class = Disease
age <= 53.5 gini = 0.476 samples = 118 value = [115, 74] class = No Disease
True
False
cholestrol <= 272.0 gini = 0.362 samples = 56 value = [77, 24] class = No Disease
BP <= 167.0 gini = 0.491 samples = 62 value = [38, 50] class = Disease
cholestrol <= 172.0 gini = 0.278 samples = 46 value = [70, 14] class = No Disease
cholestrol <= 305.5 gini = 0.484 samples = 10 value = [7, 10] class = Disease
cholestrol <= 293.5 gini = 0.498 samples = 57 value = [38, 43] class = Disease
gini = 0.0 samples = 5 value = [0, 7] class = Disease
gini = 0.32 samples = 2 value = [1, 4] class = Disease
gini = 0.221 samples = 44 value = [69, 10] class = No Disease
gini = 0.165 samples = 7 value = [1, 10] class = Disease
gini = 0.0 samples = 3 value = [6, 0] class = No Disease
gini = 0.469 samples = 47 value = [24, 40] class = Disease
gini = 0.291 samples = 10 value = [14, 3] class = No Disease

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	cholestrol	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.