upGrad

SUMMARY

Ensembles and Random Forest

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

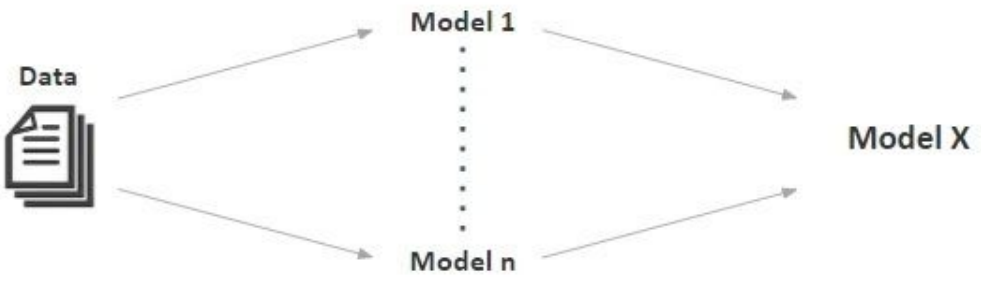
A random forest algorithm combines multiple decision trees to generate the final results. This process of combining more than one model to take the final decision is termed as ensemble learning.

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, the most popular model 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 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

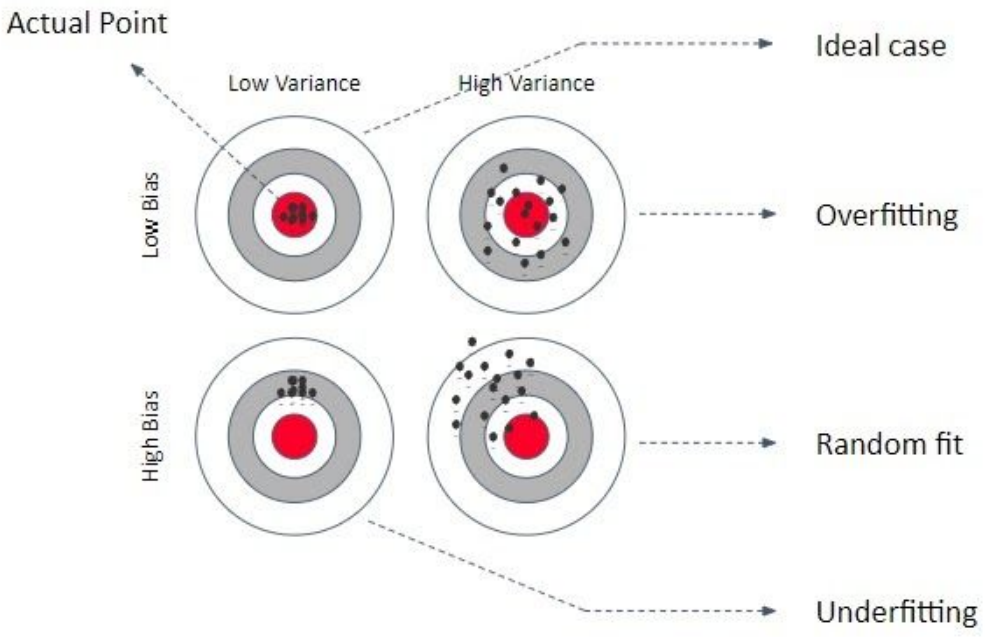
Ensemble vs Single Model

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 generalisable 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.

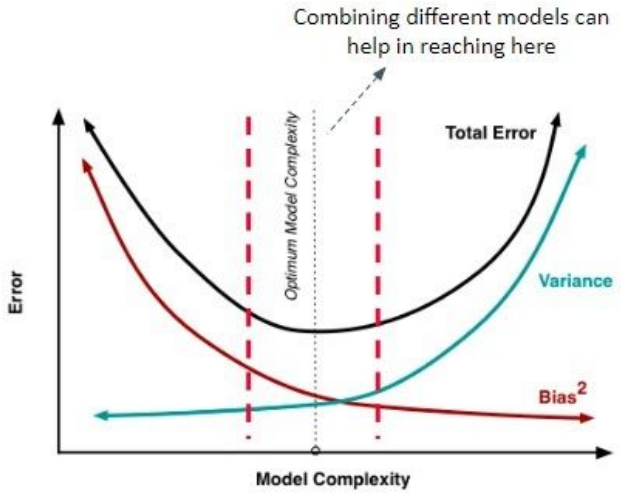
Now, you are aware of some of the shortcomings of using a single model. However, it is also essential to understand how multiple models come together to solve the problems present in a single model. The model that works extremely well on the training set (low bias) may not be generalisable (high variance),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



Bias Variance Trade-Off

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, then the probability that you will end up with a model with a low bias and a low variance is minimal, as 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 (low bias) and the other that helps to generalise the results over the unseen data (low variance). As a result, the combination of these two can provide a balanced model (with a low bias and a low variance) 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

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. This classifier assigns a weight over the output of every base model. In this way, the individual models are combined with different weights to obtain the final prediction.

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

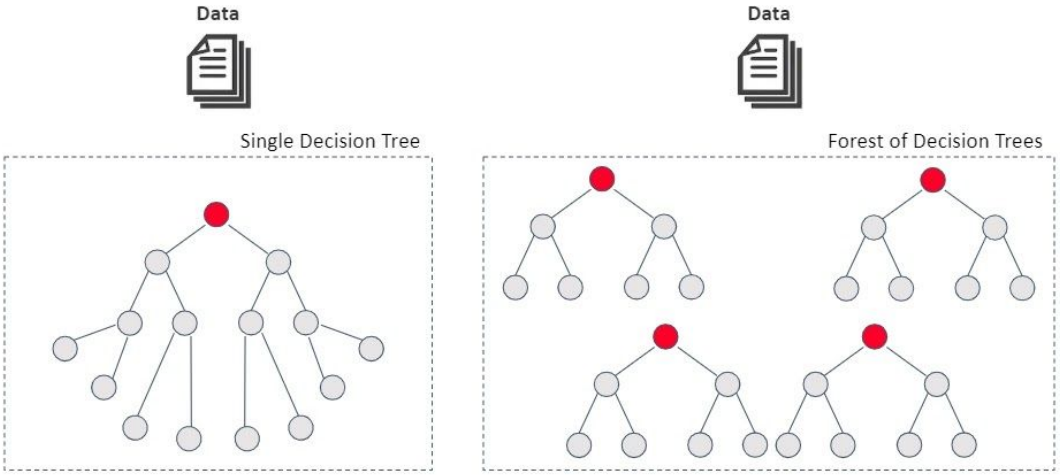
The random forest algorithm is an ensemble of decision trees that uses bagging to generate different base models. So far, random forest has been the most successful model among the bagging ensembles.

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.

Aggregation implies combining the results of different models present in the ensemble. You must understand that bagging is a technique in itself and is not specific to random forests.

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.

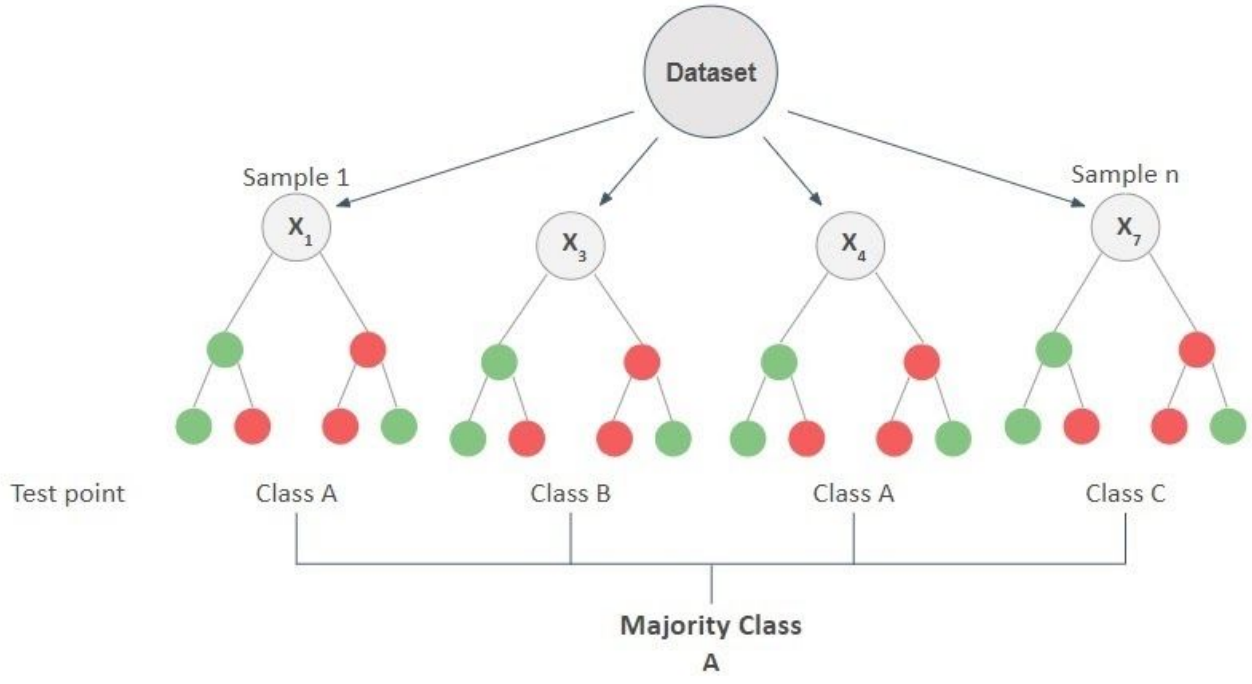
It takes a majority vote on the predictions made by models for every data point and provides the output as the class that has received the highest number of votes. In the case of regression, it will be the average of all the predicted values available for a particular data point.



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.



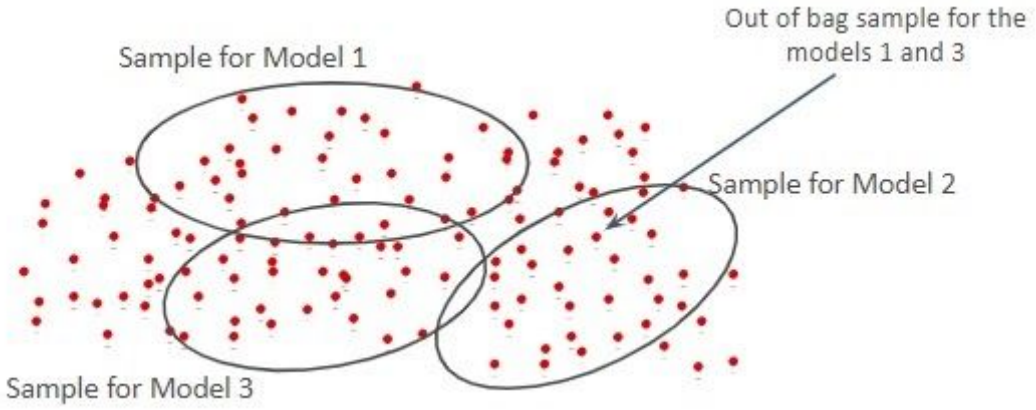
Random Forest Algorithm

upGrad

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 model is evaluated by checking the accuracy of every base tree on their respective OOB samples. In other words, the OOB score is the mean prediction accuracy on each training point X_i using only the trees that do not have X_i in their bootstrapped sample used for building the model.

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

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.

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

You know that the OOB error is beneficial when your data set is small. Though the OOB error is on unseen data, there is a disadvantage, as you use the entire training data set to calculate the OOB error. In other words, if the test data set differs from the training data set that is completely unseen, then the model may not perform well. Hence, when the data set is large, you should definitely perform cross-validation, preferably k-fold cross-validation.

Let's recall the various factors that affect the performance of SageMaker. The factors that could affect the performance of a random forest model are as follows:

- **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.

- **Weightage of base models in the final prediction based on performance (Stacking)**
  The performance of the final model can be enhanced further by running the base models through another classifier/regressor (like in stacking). 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. However, this can cause the problem of feature dominance and, hence, should be used with a check as it can lead to overfitting.

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. Variety
2. Stability
3. Immunity to the curse of dimensionality
4. Parallelisation

The disadvantages of the random forest model are as follows:

1. Lack of interpretability
2. High computational costs

The industrial applications of the random forest model are as follows:

1. Medicine
2. Motion sensing
3. Finance
4. Banking
5. Astronomy

upGrad

Summary

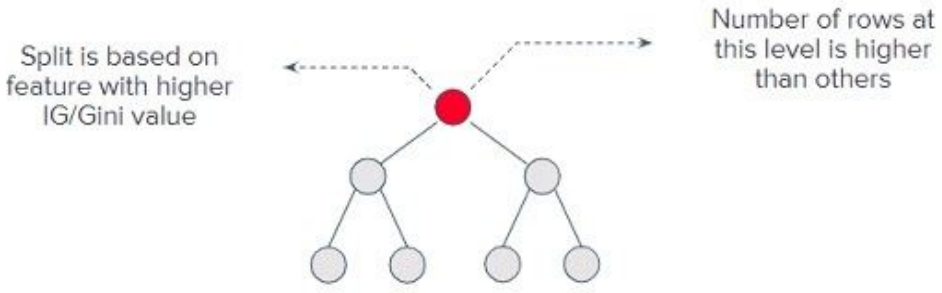
Implementing Decision Trees

In this session, you learnt about variable importance, data handling, dealing with missing values, outliers and time dependency on hyperparameters.

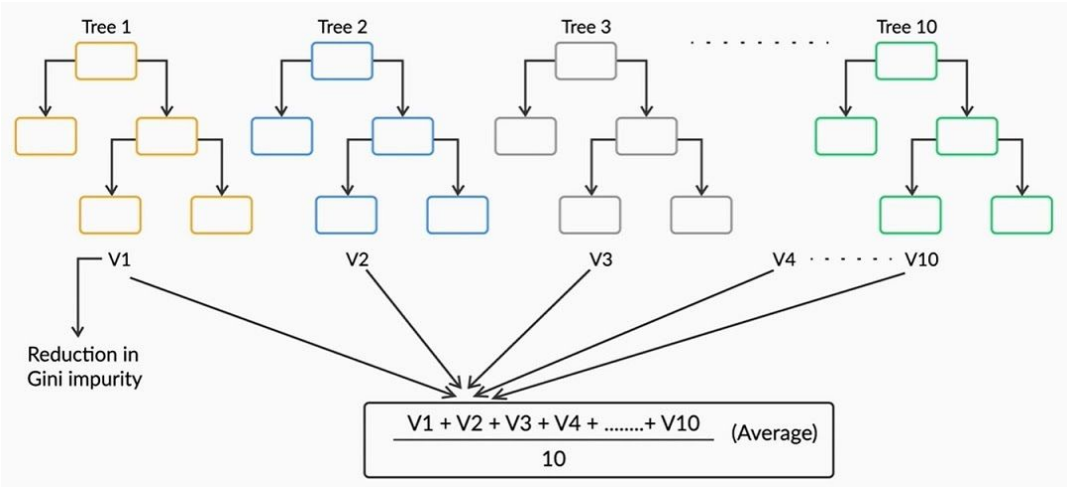
Feature Importance

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.



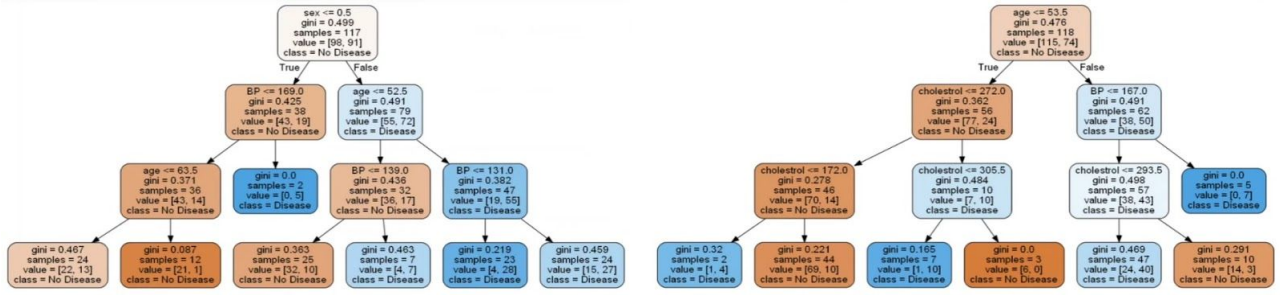
For each variable, the sum of the information gain across every tree of the forest is accumulated every time a variable is chosen to split a node. The value is then divided by the number of base trees in which the feature is involved to give the final average value.

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 a 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 model accuracy increased to 80% after you used the following hyperparameters. 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.

upGrad

Ensembles in Python

In this segment, our aims were the following:

1. Identify the variables affecting house prices, for example, the area, the number of rooms, the number of bathrooms, etc.
2. Create a linear model that quantitatively relates house prices with variables, such as the number of rooms, the area and the number of bathrooms.

You also learnt that the ensemble does not perform well compared with a single model. This is because equal weightage is given to each of the regressors to predict the final output, which is not always a good idea. This ensemble also uses the KNN or K-nearest neighbours algorithms. Stacking identifies the weights that should be allocated to different models to obtain better prediction results. This is achieved by passing the regressors to a second-level model, and the resultant ensemble performs better than any individual model and is also more efficient than a manual ensemble.

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 can split the data in a ratio of 70:30, although this is not necessary. 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.

Data Handling

Previously, you learnt that random forests are capable of handling missing values. In this segment, you learnt how they achieve that.

So far, you have dealt with missing values by either dropping them or imputing them with mean, mode or median values. However, a random forest model can overcome the missing values using inherent features. Random forests use the concept of **proximity matrix**. Here, the following is expected:

1. First, a random forest model is built using the general imputation methods such as the maximum count and average.
2. Once the first random forest model is built on the data set with a number of decision trees, it generates a proximity matrix.
3. This matrix is a nxn matrix that records the frequency of the occurrence of two data points together in a leaf node of all the trees divided by the total number of trees in the forest.

Proximity score(i,j) = Frequency of i and j appearing together / Number of trees

where, i and j are two data points from the data set.

The score can be used to measure the similarity between data points, as two points will end up in the same leaf node only if they have similar attributes. Higher the proximity score, higher are the chances that two observations have identical features. Random forest uses an **iterative method** to replace the missing values. The proximity matrix can be used to impute the missing values in both the training and testing set.

Leaf Node	Observations
1	1, 3
2	4, 5, 6
3	2, 7
4	8
5	2, 4
6	1, 7
7	5, 6
8	3, 8

	1	2	3	4	5	6	7	8
1	-	0	1/2	0	0	0	1/2	0
2	0	-	0	1/2	0	0	1/2	0
3	1/2	0	-	0	0	0	0	1/2
4	0	1/2	0	-	1/2	1/2	0	0
5	0	0	0	1/2	-	1	0	0
6	0	0	0	1/2	1	-	0	0
7	1/2	1/2	0	0	0	0	-	0
8	0	0	1/2	0	0	0	0	-

The entire process is summarised as follows:

- Training set:
    1. The missing values in the data set are first replaced by **general methods** (average, maximum count, etc.).
    2. The first iteration of the random forest model is built on the data set after the aforementioned step, which generates the first proximity matrix.
    3. For further iterations, the following is performed:
        - The proximity matrix is used to replace the missing values in the data set again. The new values are computed as the weighted average of the non-missing values with the proximity score as the weight.
        - The random forest model is built on the new data set again, which results in the new proximity matrix.

These steps are repeated until satisfactory results are obtained. Generally, it is recommended that you run 4–6 iterations to reduce the impact of missing values.

- Test set: The value of the target variable is absent along with other features.
    1. Since the target variable is absent in the test data, the observation is replicated with a copy for each class label. For example, if it is a binomial problem (0/1), then the data point will have two copies, where one point has a value of 0 and the other has a value of 1. This is done because you will require a majority vote to calculate the final prediction value.
    2. The random forest model is then built with imputed values. The observation that gets the maximum votes is selected to impute the final label. Please note that this process is performed for **imputation and not for final prediction**.

This is how random forests can efficiently deal with missing values. Another problem that is faced during data preparation is outliers. The process used to build the random forest inherently takes care of the outlier values. Owing to random sampling, the impact of outliers is reduced to a minimum, as every observation will not appear in the training set of every decision tree. Moreover, since decision trees are not sensitive to outliers, random forest, being an ensemble of decision trees, is also not sensitive to outliers.

Time Dependency

By now, you have worked on multiple case studies using random forest models. One of your key observations is that the performance of a random forest model increases with the increase in the number of trees. However, this has a limitation. As you increase the number of trees, the construction and evaluation of multiple trees result in a higher computation time. This limitation can restrict you to build a solution that requires quick analysis, as in the case of predicting stock prices. For such cases, you must understand how the computation time varies with different hyperparameters. Consider a data set with m features and n observations. Each of the n observations is represented as the vector $X_j=\{x_{j1},x_{j2},x_{j3},...,x_{jm}\}$ with the output Y_j.

Building an ensemble requires the following steps:

1. Taking a random sample of observations, suppose j= 40% of the total observations
2. Building S trees by finding all the splits from a subsetted feature space at all nodes within each tree.

Hence, the four crucial factors that affect the model building time are as follows:

- Number of points in the training set: n*j , where j = % of the data set for the training set

- Number of splitting features (by default) = $\sqrt{m}$
- Number of trees = S
- Number of levels in each tree (default, considering a binary tree) = log(n*j)

All these factors are positively correlated with the model building time. If you combine all of them, then you will obtain the following result: **Time $\propto$ S*($\sqrt{m}$*(n*j)*log(n*j)).**

upGrad

Summary

Implementing Decision Trees

In this session, you learnt how random forests work in Spark and solved case studies on the same. You also learnt how optimisation of random forests occurs in Spark.

Chicago Crime Case Study

The Chicago crime case study is based on the crime record maintained by the Chicago Police Department. It is a detailed data set that stores the information associated with all the crimes that occur in the city (except murders for which data exists for each victim). The actual data set contains records from 2001 until today. You will be taking a part of this data set to build a simple recommendation model based on the random forest model.

The data set involves multiple attributes associated with a crime such as the location, the type of crime and its description. Every crime also has an FBI code attached to it, which indicates the crime classification as outlined in The FBI's National Incident-Based Reporting System (NIBRS). This case study aims to recommend an FBI code based on the different attributes of that crime. This could help in the automatic filling of the FBI code in the police records whenever a new crime is reported. The steps followed are summarised below:

1. One of the first steps when working with big data is to clean the given data to include only the relevant information. It makes the entire process simple and effective, as unwanted details will not be loaded in the Spark memory. Therefore, we will start by dropping the columns that are not relevant to the final model or capture similar characteristics (correlated).

2. Some of the redundant columns can be identified using the data description such as 'Case number' and 'updated on' among others that do not add any significant information. Therefore, it is necessary that you drop them as the first step for reducing the processing overhead. Moreover, you also dropped multiple columns associated with the location and description of the crime and retained the x-coordinate and the y-coordinate. These variables provided the same information and, hence, can be removed from the model building process.

3. The next step of the cleaning process involves dealing with null values. You should always check for null values in the data set and then take an appropriate step to deal with them based on the scenario. The data set had a total of 37,083 null values. It is a small proportion of the data set, and hence, the corresponding rows can be dropped. However, you can run an additional check to assess whether the null values are associated with any particular class of the target variable or not. In that case, removing the null values will not be ideal.

4. Now that the data is clean, the next step is to perform feature engineering over the existing columns to extract the relevant information for model building purposes. The column 'Date' registers the date and time of the crime. Since you did not infer the schema from the data set, it was loaded in the dataframe in the string type. To extract any date or time feature, you must first convert its data type from string to timestamp.

5. Then, you can easily extract the required features. Here, you only extracted the hour during which a crime occurred. However, you can also experiment with other attributes such as the month, the date and the day

of the week.

6. Next, separate the categorical and continuous variables in two independent lists. You will use these lists to perform transformations on the data.

7. You converted the string values to indexes and created a vector of all the columns/features that you will use to make the random forest model. The generated vector is stored in the column 'features' and target variable in column 'label' of the transformed dataframe.

8. Next, you performed the following:
    - Train-test split
    - Defined a model with a set of defined hyperparameters
    - Fitted the model over the training set

9. The model provided 78% accuracy on the test set. However, it was iterated previously that the classification model should not be gauged only with the accuracy metric. This becomes even more important when you have more than one class to be predicted.

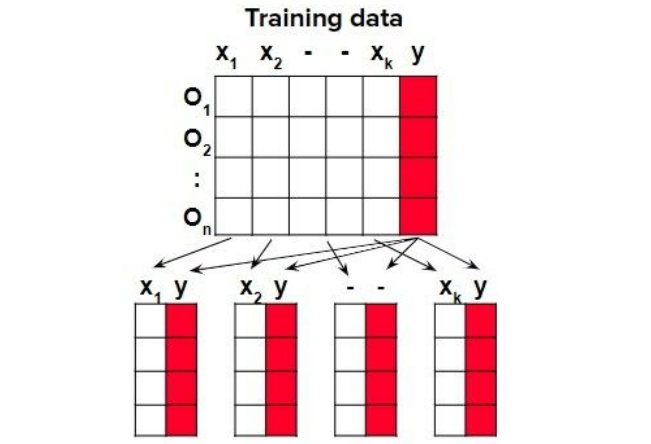
Optimisation in Spark

You have gained an understanding of the theoretical and practical concepts of random forest algorithms. You learnt how to build a basic model in the PySpark environment. However, it was emphasised repeatedly that a model must be optimised before it is implemented in a distributed environment. The two main ways to optimise it in Spark are as follows:

Data-parallel optimisation

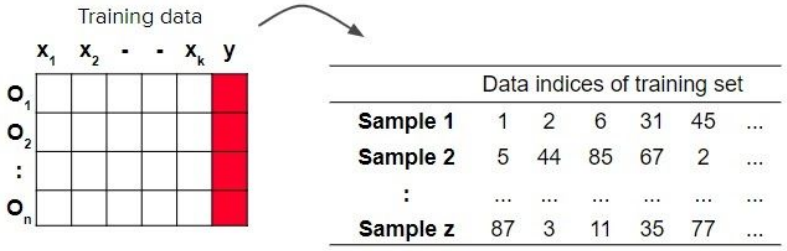
This optimisation stems from how data can be stored for processing in the Spark environment. The two methods as mentioned below:

- **Vertical data partitioning**
  This method exploits the feature of parallelism associated with the tree models. Here, the data set is split into smaller partitions that store one independent variable along with the target variable. Since the computation of information gain or the Gini index for one variable is independent of one another, the data can be split and shared across multiple executors for parallel computation. It can prove to be highly useful because it helps in parallel computation and prevents numerous copies of the entire data for deciding the splitting criteria.



Information gain or Gini importance for each variable

- **Data multiplexing**
  Data multiplexing helps you to optimise the use of Spark memory by preventing multiple copies of the entire data set for computation over bootstrapped samples. It creates a metadata associated with the samples and stores the index of the rows associated with each sample in the form of a data **sampling index (DSI)** table. The entire data is broadcasted, and then, Spark can use the DSI table to fetch only the required rows for building the decision tree on the bootstrapped sample.

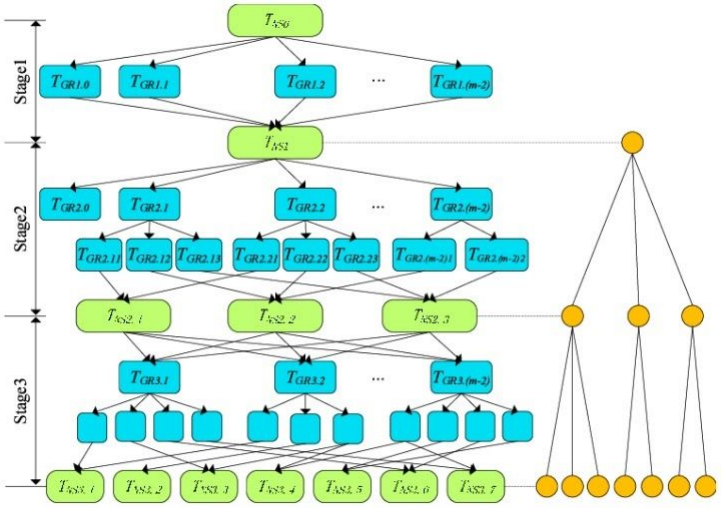


	Data indices of training set					
Sample 1	1	2	6	31	45	...
Sample 2	5	44	85	67	2	...
:	...	...	...	...	...	...
Sample z	87	3	11	35	77	...

Task-parallel optimisation

The algorithm in Spark can be parallelised at the following two levels:

- **Tree-level**
  The computation of each tree in the algorithm is independent of the other. The data associated with each tree can be distributed over different executors and processed parallelly. Higher the parallelism, more optimised is the execution. Once you build all the trees, the results from each tree can be sent to the driver to combine and obtain the final result.
- **Node-level**
  As mentioned in data-parallel optimisation, the process of computation of information gain or the Gini index for a feature to decide the split criteria for a node is independent of another. This feature of the algorithm can be exploited by running a parallel computation for every attribute after the process of vertical data partitioning. It can be further optimised by running parallel computation for every node at the same level in the tree. The image given below summarises the entire process for a single tree:



Source: Research Gate

All the optimisations mentioned in the segment have resulted in a higher performance of the model in the Spark environment. As mentioned in the video, they are part of the '**Parallel Random Forest**' (**PRF**) algorithm.