Decision Trees

In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making

As the name goes, it uses a tree-like model of decisions.

What is a Decision Tree ? How does it work ?

Decision tree is a type of supervised learning algorithm (having a predefined target variable) that is mostly used in classification problems.
It works for both categorical and continuous input and output variables.
In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on the most significant splitter / differentiator in input variables.

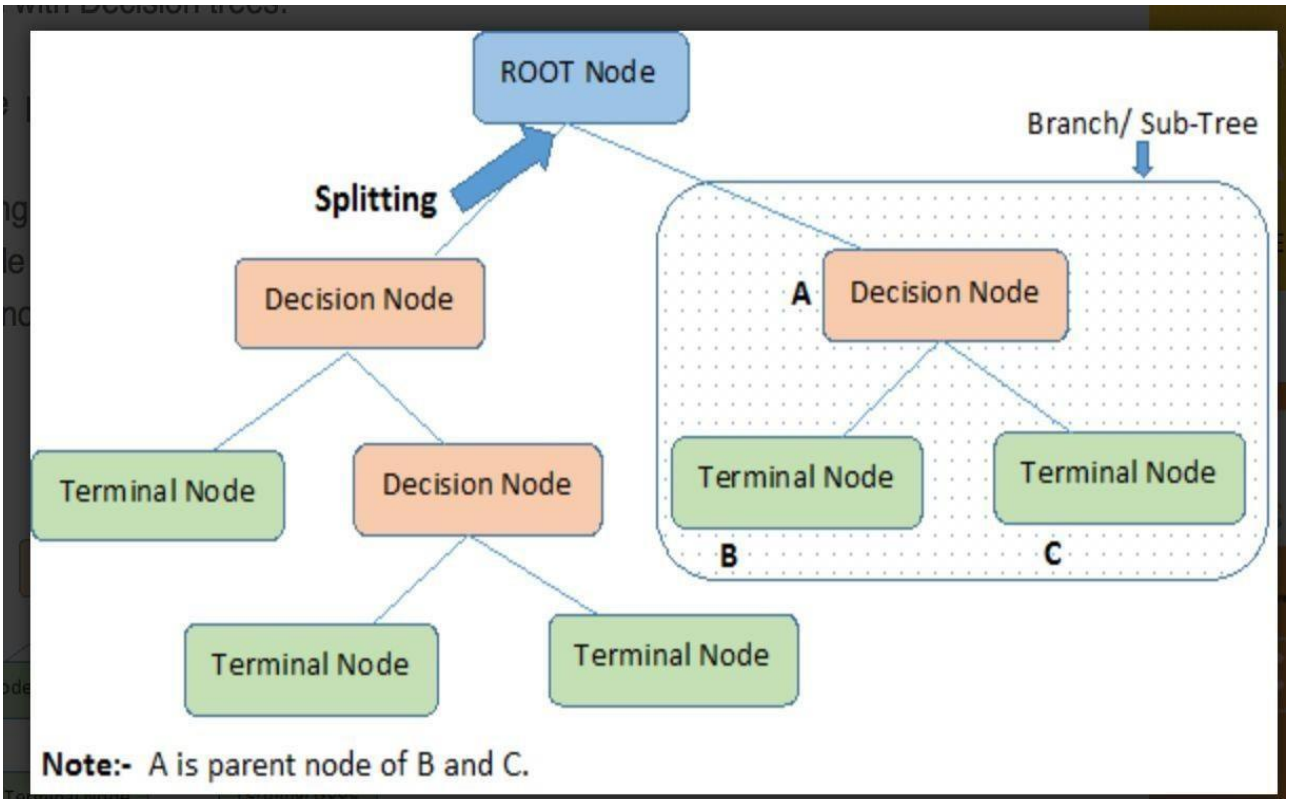
Types of Decision trees

1. Categorical Variable Decision Tree:

Decision Tree which has categorical target variable then it is called as categorical variable decision tree. Example:- In previous scenario of student problem, where the target variable was “Student will play cricket or not” i.e. YES or NO.

2. Continuous Variable Decision Tree:

Decision Tree has continuous target variable then it is called as Continuous Variable Decision Tree.



Common Algorithms used in Decision Trees:

1. CART (Classification and Regression Trees): Uses Gini Impurity/Gini as splitting criteria.
2. C4.5 (successor of ID3): Uses Information gain/Entropy as splitting Criteria.
3. ID3 (Iterative Dichotomiser 3): Uses Entropy as Splitting Criteria.
4. CHAID (Chi-square automatic interaction detection): Uses Chi-Square as splitting criteria.

Advantages and Disadvantages of Decision Trees

Advantages

- Easy to Understand
- Useful in Data exploration
- Less data cleaning required
- Data type is not a constraint

- Non Parametric Method

Disadvantages

- Over fitting
- Not fit for continuous values

Regression Trees vs Classification Trees

1. Regression trees are used when the dependent variable is continuous. Classification trees are used when the dependent variable is categorical.
2. In the case of a regression tree, the value obtained by terminal nodes in the training data is the mean response of observation falling in that region. Thus, if an unseen data observation falls in that region, we'll make its prediction with mean value.
3. In the case of classification trees, the value (class) obtained by the terminal node in the training data is the mode of observations falling in that region. Thus, if an unseen data observation falls in that region, we'll make its prediction with mode value.

How does a tree decide where to split?

The collection of split points examined for every variable is determined by whether the variable is numeric or categorical. The values of the variable taken by the cases at that node also play a role.
When a **predictor is numeric**, and all values are unique, there are **n – 1** split points for n data points. Because this can be a high number, it is typical to evaluate just split points at specific percentiles of the value distribution. For example, we could look at every tenth percentile (that is, 10 percent , 20 percent , 30 percent , etc).
When a **predictor is categorical**, we can choose whether to split it into one child node per class (multiway splits) or merely two child nodes (binary split). The Root split is multiway in the diagram above. Because multiway splits fragment the data into small parts too quickly, binary splits are usually used. This results in a bias toward dividing predictors with numerous classes since they are more likely to yield relatively pure child nodes, resulting in overfitting.

If a categorical predictor has only two classes, there is only one feasible split. However, if a categorical predictor includes more than two classes, other conditions may apply.

If there are only a few classes, all possible splits into two child nodes can be explored. There are 2k – 1 – 1 splits for k classes, which is computationally prohibitive if k is a large number.

If there are multiple classes, they can be sorted by their average output value. We can then divide the ordered classes into two groups using a binary split. This means that given k classes, there are k – 1 potential splits.
There are more splits to consider if k is large. As a result, there is a larger likelihood that a certain split will result in a significant improvement and is thus preferable. As a result, trees are biassed toward separating variables with numerous classes over variables with fewer classes.

Overfitting in decision trees and how to avoid it?

Overfitting is one of the key challenges faced while modeling decision trees. If there is no limit set of a decision tree, it will give you 100% accuracy on the training set because in the worst case it will end up making 1 leaf for each observation. Thus, preventing overfitting is pivotal while modeling a decision tree and it can be done in 2 ways:

1. Setting constraints on tree size
2. Tree pruning

Setting constraints on tree size:

1. Minimum samples for a node split

- Defines the minimum number of samples (or observations) which are required in a node to be considered for splitting.
- Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.
- Too high values can lead to under-fitting hence, it should be tuned using CV.

2. Minimum samples for a terminal node (leaf)

- Defines the minimum samples (or observations) required in a terminal node or leaf.
- Used to control over-fitting similar to min_samples_split.
- Generally lower values should be chosen for imbalanced class problems because the regions in which the minority class will be in majority will be very small.

3. Maximum depth of tree (vertical depth)

- The maximum depth of a tree.
- Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.
- Should be tuned using CV.

4.Maximum number of terminal nodes

- The maximum number of terminal nodes or leaves in a tree.
- Can be defined in place of max_depth. Since binary trees are created, a depth of ‘n’ would produce a maximum of 2^n leaves.

5.Maximum features to consider for split

- The number of features to consider while searching for a best split. These will be randomly selected.
- As a thumb-rule, square root of the total number of features works great but we should check upto 30-40% of the total number of features.
- Higher values can lead to overfitting but it depends on case to case.

Pruning

Growing the tree beyond a certain level of complexity leads to overfitting.
In our data, age doesn’t have any impact on the target variable.
Growing the tree beyond Gender is not going to add any value. Need to cut it at Gender.
This process of trimming trees is called Pruning.

Pruning to Avoid Overfitting

- Pruning helps us to avoid overfitting
- Generally it is preferred to have a simple model, it avoids overfitting issue
- Any additional split that does not add significant value is not worthwhile.
- We can avoid overfitting by changing the parameters like
    - max_leaf_nodes
    - min_samples_leaf
    - max_depth

Pruning Parameters

max_leaf_nodes

- Reduce the number of leaf nodes

min_samples_leaf

- Restrict the size of sample leaf

- Minimum sample size in terminal nodes can be fixed to 30, 100, 300 or 5% of total

max_depth

- Reduce the depth of the tree to build a generalized tree.
- Set the depth of the tree to 3, 5, 10 depending after verification on test data

Code-Tree Pruning

```
#We will rebuild a new tree by using above data and see how it works by tweeking the parameteres

dtree = tree.DecisionTreeClassifier(criterion = "gini", splitter = 'random', max_leaf_nodes = 10, min_sa
dtree.fit(X_train,y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
            max_features=None, max_leaf_nodes=10, min_samples_leaf=5,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='random')
```

```
predict3 = dtree.predict(X_train)
print(predict3)
```

```
[1 1 0 0 0 1 1 1 1 0 0 1 0 0]
```

```
predict4 = dtree.predict(X_test)
print(predict4)
```

```
[1 1 0 0 0 1]
```

```
#Accuracy of the model that we created with modified model parameters.
score2 = dtree.score(X_test, y_test)
score2
```

```
0.83333333333333337/
```

Greedy Approach

Greedy Approach is based on the concept of Heuristic Problem Solving by making an optimal local choice at each node. By making these local optimal choices, we reach the approximate optimal solution globally."
The algorithm can be summarized as :
1. At each stage (node), pick out the best feature as the test condition.
2. Now split the node into the possible outcomes (internal nodes).
3. Repeat the above steps until all the test conditions have been exhausted into leaf nodes.
When you start to implement the algorithm, the first question is: 'How to pick the starting test condition?'
To make that decision, you would use the concept of splitting.

Linear models VS Decision trees

"If I can use logistic regression for classification problems and linear regression for regression problems, why is there a need to use trees"? Many of us have this question. And, this is a valid one too.
Actually, you can use any algorithm. It is dependent on the type of problem you are solving.
Let's look at some key factors which will help you to decide which algorithm to use:

1. If the relationship between dependent & independent variables is well approximated by a linear model, linear regression will outperform tree based models.
2. If there is a high nonlinearity & complex relationship between dependent & independent variables, a tree model will outperform a classical regression method.
3. If you need to build a model which is easy to explain to people, a decision tree model will always do better than a linear model. Decision tree models are even simpler to interpret than linear regression!

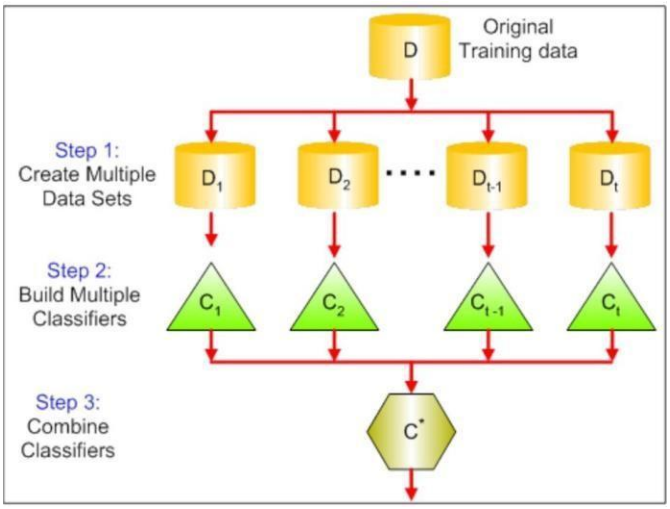
What are ensemble methods in tree based modeling ?

The literary meaning of the word 'ensemble' is group. Ensemble methods involve a group of predictive models to achieve better accuracy and model stability. Ensemble methods are known to impart supreme boost to tree based models.
Like every other model, a tree based model also suffers from the plague of bias and variance. Bias means, 'how much on an average are the predicted values different from the actual value.' Variance means, 'how different will the predictions of the model be at the same point if different samples are taken from the same population'.

You build a small tree and you will get a model with low variance and high bias. How do you manage to balance the trade-off between bias and variance ?
Normally, as you increase the complexity of your model, you will see a reduction in prediction error due to lower bias in the model. As you continue to make your model more complex, you end up overfitting your model and your model will start suffering from high variance.

What is Bagging? How does it work?

Bagging is a technique used to reduce the variance of our predictions by combining the result of multiple classifiers modeled on different sub-samples of the same data set. The following figure will make it clearer:



The steps followed in bagging are:

1. Create Multiple Data Sets:
   - Sampling is done with replacement on the original data and new datasets are formed.
   - The new data sets can have a fraction of the columns as well as rows, which are generally hyper-parameters in a bagging model
   - Taking row and column fractions less than 1 helps in making robust models, less prone to overfitting
2. Build Multiple Classifiers:
   - Classifiers are built on each data set.
   - Generally the same classifier is modeled on each data set and predictions are made.
3. Combine Classifiers:
   - The predictions of all the classifiers are combined using a mean, median or mode value depending on the problem at hand.

- The combined values are generally more robust than a single model.

Bagging

Partitioning of data	Random
Goal to achieve	Minimum variance
Methods used	Random subspace
Functions to combine single model	Weighted average
Example	Random Forest

Definition:

Bagging is used when the goal is to reduce the variance of a decision tree classifier. Here the objective is to create several subsets of data from training samples chosen randomly with replacement. Each collection of subset data is used to train their decision trees. As a result, we get an ensemble of different models. Average of all the predictions from different trees are used which is more robust than a single decision tree classifier.

Bagging Steps:

- Suppose there are N observations and M features in the training data set. A sample from the training data set is taken randomly with replacement.
- A subset of M features are selected randomly and whichever feature gives the best split is used to split the node iteratively.
- The tree has grown to the largest.
- Above steps are repeated n times and prediction is given based on the aggregation of predictions from n number of trees.

Advantages:

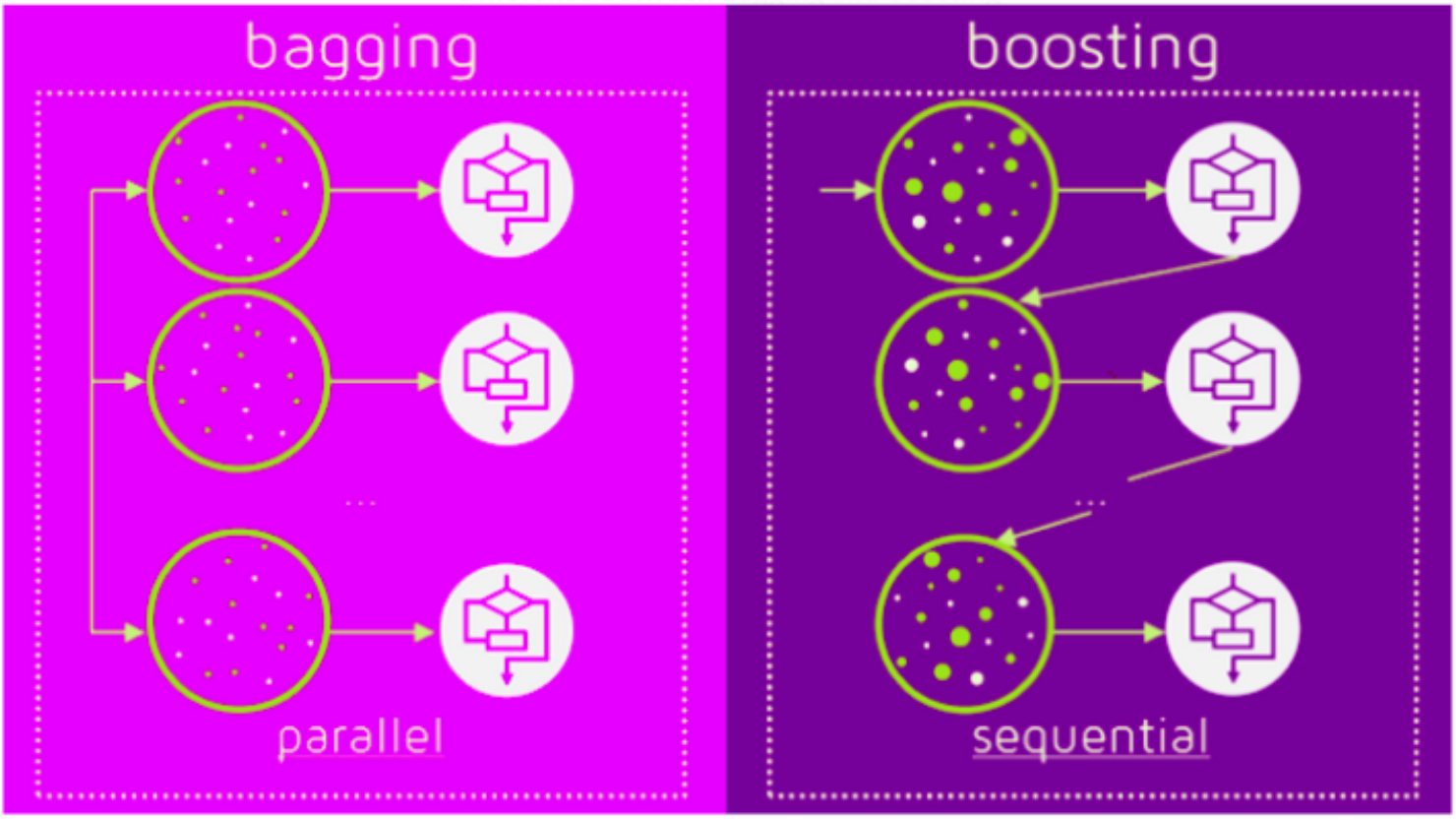
- Reduces overfitting of the model.
- Handles higher dimensionality data very well.
- Maintains accuracy for missing data.

Disadvantages:

- Since final prediction is based on the mean predictions from subset trees, it won't give precise values for the classification and regression model.

Python Syntax:

- rfm = RandomForestClassifier(n_estimators=80, oob_score=True, n_jobs=-1, random_state=101, max_features = 0.50, min_samples_leaf = 5)
- fit(x_train, y_train)
- predicted = rfm.predict_proba(x_test)



Boosting

Partitioning of data	Higher vote to misclassified samples
Goal to achieve	Increase accuracy
Methods used	Gradient descent
Functions to combine single model	Weighted majority vote
Example	Ada Boost

Definition:

Boosting is used to create a collection of predictors. In this technique, learners are learned sequentially with early learners fitting simple models to the data and then analysing data for errors. Consecutive trees (random sample) are fit and at every step, the goal is to improve the accuracy from the prior tree. When an input is misclassified by a hypothesis, its weight is increased so that the next hypothesis is more likely to classify it correctly. This process converts weak learners into better performing models.

Boosting Steps:

- Draw a random subset of training samples d1 without replacement from the training set D to train a weak learner C1
- Draw second random training subset d2 without replacement from the training set and add 50 percent of the samples that were previously falsely classified/misclassified to train a weak learner C2
- Find the training samples d3 in the training set D on which C1 and C2 disagree to train a third weak learner C3
- Combine all the weak learners via majority voting.

Advantages:

- Supports different loss functions (we have used ‘binary:logistic’ for this example).
- Works well with interactions.

Disadvantages:

- Prone to overfitting.
- Requires careful tuning of different hyper-parameters.

Python Syntax:

- from xgboost import XGBClassifier
- xgb = XGBClassifier(objective=’binary:logistic’, n_estimators=70, seed=101)
- fit(x_train, y_train)
- predicted = xgb.predict_proba(x_test)

Random Forest

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

Feature Importance

Another great quality of the random forest algorithm is that it is very easy to measure the relative importance of each feature on the prediction. Sklearn provides a great tool for this that measures a feature's importance by looking at how much the tree nodes that use that feature reduce impurity across all trees in the forest. It computes this score automatically for each feature after training and scales the results so the sum of all importance is equal to one.
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.

```
importances = pd.DataFrame({'feature':X_train.columns,'importance':np.round
(random_forest.feature_importances_,3)})
importances = importances.sort_values('importance',ascending=False).set_ind
ex('feature')
```

```
In [50]: importances.head(15)
```

Out[50]:

	importance
feature	
Title	0.202
Sex	0.167
Age_Class	0.092
Deck	0.091
Pclass	0.079
Age	0.076
Fare	0.071
relatives	0.055
Embarked	0.053
Fare_Per_Person	0.040
SibSp	0.039
Parch	0.024
not_alone	0.010

Hyperparameter Tuning

Hyperparameter Tuning In RF

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.

1. Increasing the predictive power

- Firstly, there is the n_estimators hyperparameter, which is just the number of trees the algorithm builds before taking the maximum voting or taking the averages of predictions. In general, a higher number of trees increases the performance and makes the predictions more stable, but it also slows down the computation.
- Another important hyperparameter is max_features, which is the maximum number of features random forest considers to split a node. Sklearn provides several options, all described in the Documentation.

- The last important hyperparameter is min_sample_leaf. This determines the minimum number of leaves required to split an internal node.

2. Increasing the model's speed

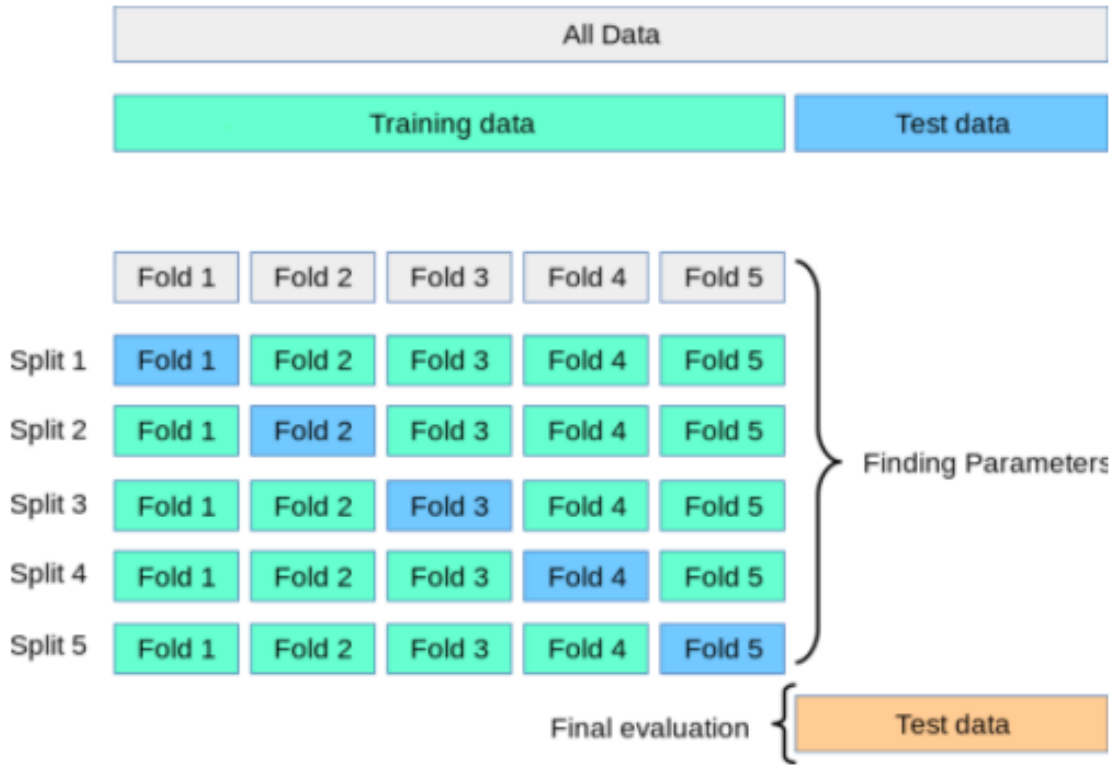
- The n_jobs hyperparameter tells the engine how many processors it is allowed to use. If it has a value of one, it can only use one processor. A value of “-1” means that there is no limit.

- The random_state hyperparameter makes the model’s output replicable. The model will always produce the same results when it has a definite value of random_state and if it has been given the same hyperparameters and the same training data.

- Lastly, there is the oob_score (also called oob sampling), which is a random forest cross-validation method. In this sampling, about one-third of the data is not used to train the model and can be used to evaluate its performance. These samples are called the out-of-bag samples. It's very similar to the leave-one-out-cross-validation method, but almost no additional computational burden goes along with it.

Cross Validation

Cross Validation is a technique which involves reserving a particular sample of a dataset on which you do not train the model. Later, you test your model on this sample before finalizing it.



K-Fold is popular and easy to understand, it generally results in a less biased model compared to other methods. Because it ensures that every observation from the original dataset has the chance of appearing in the training and test set. This is one among the best approaches if we have limited input data. This method follows the below steps.

1. Split the entire data randomly into k folds (the value of k shouldn't be too small or too high, ideally we choose 5 to 10 depending on the data size). The higher value of K leads to a less biased model (but large variance might lead to overfit), whereas the lower value of K is similar to the train-test split approach we saw before.
2. Then fit the model using the K — 1 (K minus 1) folds and validate the model using the remaining Kth fold. Note down the scores/errors.
3. Repeat this process until every K-fold serves as the test set. Then take the average of your recorded scores. That will be the performance metric for the model.

Decision Tree v/s Random Forest:

When using a decision tree model on a given training dataset the accuracy keeps improving with more and more splits. You can easily overfit the data and don't know when you have crossed the line unless you are using cross validation (on a training data set). The advantage

of a simple decision tree is that the model is easy to interpret, you know what variable and what value of that variable is used to split the data and predict outcome.
A random forest is like a black box and works as mentioned in the above answer. It's a forest you can build and control. You can specify the number of trees you want in your forest (n_estimators) and also you can specify the max number of features to be used in each tree. But you cannot control the randomness, you cannot control which feature is part of which tree in the forest, you cannot control which data point is part of which tree. Accuracy keeps increasing as you increase the number of trees, but becomes constant at certain points. Unlike decision trees, it won't create highly biased models and reduce the variance.

When to use to decision tree:

1. When you want your model to be simple and explainable
2. When you want non parametric model
3. When you don't want to worry about feature selection or regularization or worry about multicollinearity.
4. You can overfit the tree and build a model if you are sure the validation or test data set is going to be a subset of the training data set or almost overlapping instead of unexpected.

When to use random forest :

1. When you don't bother much about interpreting the model but want better accuracy.
2. Random forest will reduce variance part of error rather than bias part, so on a given training data set decision tree may be more accurate than a random forest. But on an unexpected validation data set, Random forest always wins in terms of accuracy.