Analytics Vidhya

TREE - BASED ALGORITHMS

A complete Book from scratch in

R & PYTHON

by: Analytics Vidhya

Dedicated to our thriving community – which learns, loves, shares, competes, provides feedback and owns what we have built as much as we do.

A Big Thanks to the entire Analytics Vidhya team, who built this fabulous community and continue to do so. The hard work you all put in enables me to think bigger and give shape to our journey.

Special Thanks to Sunil Ray and other authors on Analytics Vidhya- who contributed to the book! This would not be in this form without your contributions.

About the book

This book is prepared to help beginners learn tree-based algorithms from scratch. The book talks about Tree-Based algorithms like decision trees, random forest, gradient boosting in detail. After the successful completion of this book, you would be able to work on and build tree-based algorithms and predictive models.

Now, you might be wondering - is this book right for you?

The concepts we've covered in this book require a basic understanding of core machine learning concepts. It would be beneficial for you if you are aware of what machine learning is, the different types of machine learning algorithms, the difference between supervised and unsupervised problems, and so on.

The code for building tree-based algorithms is covered in both Python and R, so naturally, you would need to have a basic understanding of either programming language.

If you're a complete beginner to the world of machine learning, Python and R, we recommend the below resources to get you started:

- Python for Data Science
- Learn R for Data Science
- Introduction to Data Science
- Machine Learning Starter Program

About the Author

Analytics Vidhya is the World's Leading Data Science Community & Knowledge Portal. The mission is to create the next-gen data science ecosystem! This platform allows people to learn & advance their skills through various training programs, know more about data science from its articles, Q&A forum, and learning paths. Also, we help professionals & amateurs to sharpen their skillsets by providing a platform to participate in Hackathons. Our viewers remain updated with the latest happenings around the world of analytics using our monthly newsletters. Stay in touch with us to be a perfect and informative data practitioner. www.analyticsvidhya.com.

Our Other Platforms
Courses: https://courses.analyticsvidhya.com/
Blog: https://www.analyticsvidhya.com/blog/
DataHack: https://datahack.analyticsvidhya.com/contest/all/
Jobs: https://jobsnew.analyticsvidhya.com/jobs/all
Bootcamp: https://www.analyticsvidhya.com/data-science-immersive-bootcamp/
Initiate AI: https://initiateai.analyticsvidhya.com/
Discuss: https://discuss.analyticsvidhya.com/

Table of Contents

About the Book *iii*
About the Author *iv*

What is Machine Learning? 1
How do Machines Learn? 3
Types of Machine Learning Algorithms 5
    What are the Different Algorithms used in Machine Learning? 6
Introduction to Tree-Based Algorithms 7
Chapter 1: What is a Decision Tree? How does it work? 8
    Types of Decision Trees 9
    Important Terminology related to Tree based Algorithms 10
    Advantages 11
    Disadvantages 11
Chapter 2: Regression Trees vs Classification Trees 12
Chapter 3: How does a tree-based algorithm decide where to split? 14
    Gini 14
    Chi-Square 15
    Information Gain 17
    Reduction in Variance 18
Chapter 4: What are the key parameters of tree-based algorithms, and how can we avoid over-fitting in decision trees? 20
    Setting Constraints on tree-based algorithms 20
    Pruning in tree-based algorithms 21
Chapter 5: Are tree-based algorithms better than linear models? 24
Chapter 6: Working with tree-based algorithms Trees in R and Python 25
Chapter 7: What are ensemble methods in tree-based algorithms 29
Chapter 8: What is Bagging? How does it work? 30
Chapter 9: What is Random Forest? 32
    How does it work? 32
    Advantages of Random Forest 34
    Disadvantages of Random Forest 35

Python & R implementation 35
Chapter 10: What is Boosting? 38
How does it work? 39
Chapter 11: Which is more powerful: GBM or XGBoost? 40
Chapter12: Working with GBM in R and Python 42
GBM in R (with cross validation) 44
GBM in Python 45
Working with XGBoost in R and Python 47
Chapter 13: Where to practice? 48
Analytics Vidhya Community Discussion forums 50

What is Machine Learning?

Machine Learning is the science of teaching machines how to learn by themselves. Now, you might be thinking – why on earth would we want machines to learn by themselves? Well – it has a lot of benefits.

"Machines can do high-frequency repetitive tasks with high accuracy without getting bored."

For example – the task of mopping and cleaning the floor. When a human does the task – the quality of the outcome will vary. We get exhausted/bored after a few hours of work and the chances of getting sick also impact the outcome.

Depending on the place – it could also be hazardous or risky for a human.

On the other hand, if we can teach machines to detect whether the floor needs cleaning and mopping and how much cleaning is required based on the condition of the floor and the type of the floor – machines would perform the same job far better. They can go on to do that job without getting tired or sick!

This is what Machine Learning aims to do – enable machines to learn on their own. In order to answer the questions like:

- Whether the floor needs cleaning and mopping?
- How long does the floor need to be cleaned?

Machines need a way to think, and this is precisely where machine learning models help. The machines capture data from the environment and feed it to the machine learning model. The model then uses this data to predict things like:

- Whether the floor needs cleaning or not, or
- For how long does it need to be cleaned, and so on.

How do Machines Learn?

Sadly, things which are usually intuitive to humans can be very difficult for machines. You only need to demonstrate cleaning and mopping to a human a few times – before they can perform it on their own.

But, that is not the case with machines. We need to collect a lot of data along with the desired outcomes in order to teach machines to perform specific tasks. **This is where machine learning comes into play.**

Machine Learning would help the machine understand the kind of cleaning, the intensity of cleaning, and the duration of cleaning based on the conditions and nature of the floor.

We encourage you to read more about machine learning in-depth in the popular "Machine Learning Simplified" e-book by Analytics Vidhya.

The book aims to simplify machine learning by explaining it in simple words. It provides an overview of ML, recent developments in machine learning & Deep Learning, machine learning algorithms and current challenges in Machine Learning. Here's what you can expect:

- What is Machine Learning
- Applications of Machine Learning
- How do Machines Learn?
- Why is ML getting so much attention?
- Steps required to Build a Machine Learning Model
- How can one build a career in Machine learning?

And much, much more!

Broadly, there are 3 types of Machine Learning Algorithms

1. Supervised Learning

How it works: These algorithms consist of a target/outcome variable (or dependent variable) which is to be predicted from a given set of predictors (independent variables). Using these sets of variables, we generate a function that maps inputs to desired outputs. The training process continues until the model achieves a desired level of accuracy on the training data. Examples of Supervised Learning: Regression, Decision Tree, Random Forest, KNN, Logistic Regression etc.

Decision Trees and other tree-based models are part of supervised learning.

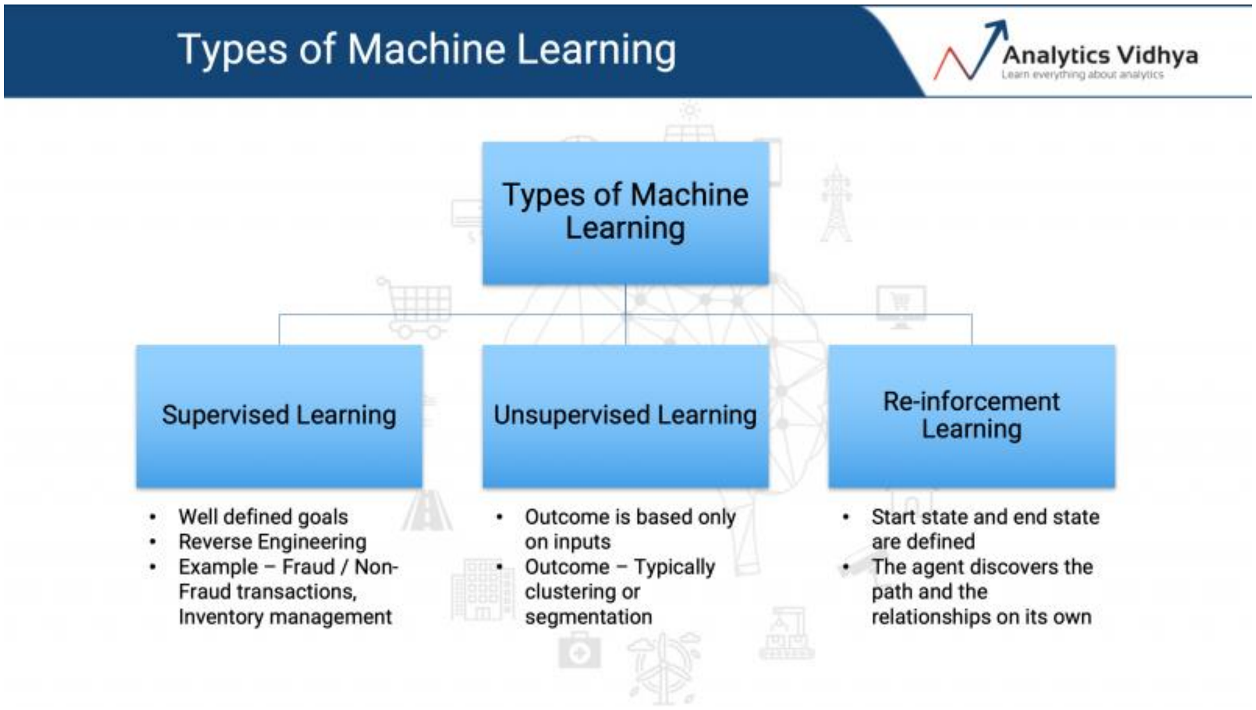
2. Unsupervised Learning

How it works: In this algorithm, we do not have any target or outcome variable to predict / estimate. It is used for clustering population in different groups, which is widely used for segmenting customers in different groups for specific intervention. Examples of Unsupervised Learning: Apriori algorithm, K-means.

3. Reinforcement Learning

How it works: Using reinforcement learning, the machine is trained to make specific decisions. It works this way: the machine is exposed to an environment where it trains itself continually using trial and error. This machine learns from past experience and tries to capture the best possible knowledge to make accurate business decisions. Example of Reinforcement Learning: Markov Decision Process.

Here is the difference between these different types of machine learning algorithms in a visual format:


Types of Machine Learning
Analytics Vidhya
Learn everything about analytics
Types of Machine Learning
Supervised Learning
Unsupervised Learning
Re-inforcement Learning
• Well defined goals
• Reverse Engineering
• Example – Fraud / Non-Fraud transactions, Inventory management
• Outcome is based only on inputs
• Outcome – Typically clustering or segmentation
• Start state and end state are defined
• The agent discovers the path and the relationships on its own

What are the Different Algorithms used in Machine Learning?

There are different types of machine learning algorithms categorized based on the problem we're trying to solve:

- Supervised Learning
    - Linear Regression
    - Logistic Regression
    - k-nearest neighbours
    - Decision Trees
    - Random Forest
    - Gradient Boosting Machines
    - XGBoost
    - Support Vector Machines (SVM)
    - Neural Networks
- Unsupervised Learning
    - k means clustering
    - Hierarchical clustering
    - Neural Network
- Reinforcement Learning

These are further divided into their different types. In this book, we are going to be focusing on the tree-based aspect of machine learning algorithms. Let's dive in!

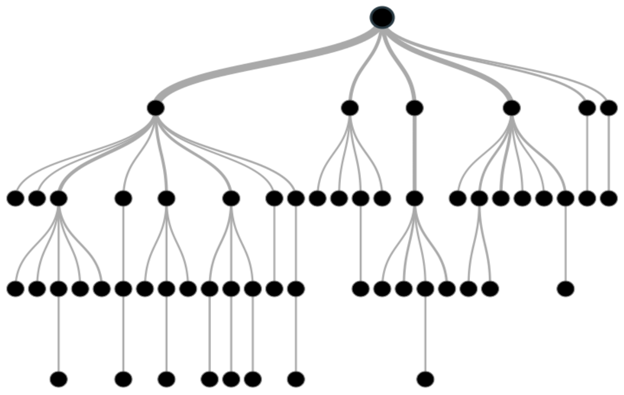
Introduction to Tree-Based Algorithms

Tree-based algorithms are considered to be one of the best and mostly used supervised learning methods. Tree-based algorithms empower predictive models with high accuracy, stability and ease of interpretation. Unlike linear models, they map non-linear relationships quite well. They are adaptable at solving any kind of problem at hand (classification or regression).

Methods like decision trees, random forest, gradient boosting are popularly used in all kinds of data science problems. Hence, for every analyst (fresher also), it’s important to learn these algorithms and use them for modelling.

Chapter 1: What is a Decision Tree? How does it work?

A 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 most significant splitter/differentiator in input variables.

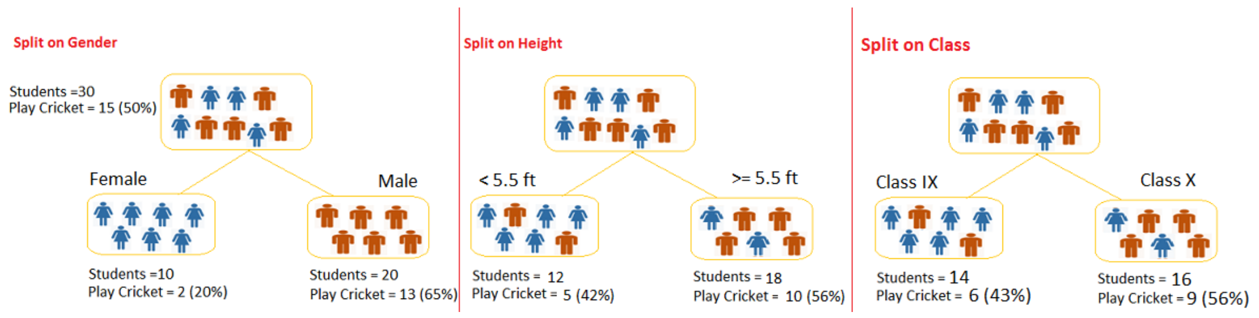


Tree-based modelling in R and Python

Example: -

Let’s say we have a sample of 30 students with three variables Gender (Boy/ Girl), Class (IX/ X) and Height (5 to 6 ft). 15 out of these, 30 play cricket in leisure time. Now, I want to create a model to predict who will play cricket during a leisure period? In this problem, we need to segregate students who play cricket in their leisure time based on highly significant input variable among all three.

This is where decision tree helps, it will segregate the students based on all values of three variable and identify the variable, which creates the best homogeneous sets of students (which are heterogeneous to each other). In the snapshot below, you can see that variable Gender is able to identify the best homogeneous sets compared to the other two variables.



As mentioned above, the decision tree identifies the most significant variable and it's value that gives the best homogeneous sets of population. Now the question which arises is, how does it identify the variable and the split? To do this, the decision tree uses various algorithms, which we will discuss in the following section.

Types of Decision Trees

Types of decision tree are based on the type of target variable we have. It can be of two types:

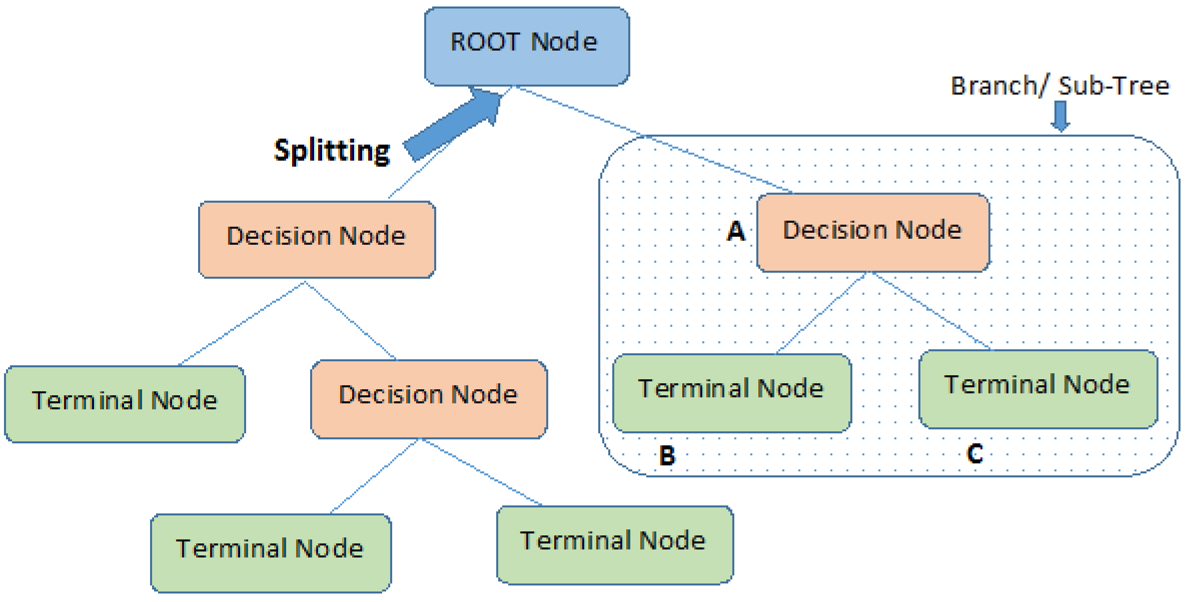
1. **Categorical Variable Decision Tree:** Decision Tree which has a categorical target variable then it called a categorical variable decision tree. Example: - In the above scenario of student problem, where the target variable was "Student will play cricket or not" i.e. YES or NO.
2. **Continuous Variable Decision Tree:** Decision Tree has continuous target variable then it is called as Continuous Variable Decision Tree.

Example: - Let's say we have a problem to predict whether a customer will pay his renewal premium with an insurance company (yes/ no). Here we know that the income of customer is a significant variable, but the insurance company does not have income details for all customers. Now, as we know this is an important variable, then we can build a decision tree to predict customer income based on occupation, product and various other variables. In this case, we are predicting values for the continuous variable.

Important Terminology related to Tree based Algorithms

Let’s look at the basic terminology used with Decision trees:

1. **Root Node:** It represents the entire population or sample, and this further gets divided into two or more homogeneous sets.
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called a decision node.
4. **Leaf/ Terminal Node:** Nodes that do not split is called Leaf or Terminal node.



Note:- A is parent node of B and C.

5. **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. You can say the opposite process of splitting.
6. **Branch / Sub-Tree:** A subsection of the entire tree is called branch or sub-tree.
7. **Parent and Child Node:** A node, which is divided into sub-nodes, is called a parent node of sub-nodes, whereas sub-nodes are the child of the parent node.

These are the terms commonly used for decision trees. As we know that every algorithm has advantages and disadvantages, below are the important factors which one should know.

Advantages

1. **Easy to Understand**: Decision tree output is very easy to understand even for people from the non-analytical background. It does not require any statistical knowledge to read and interpret them. Its graphical representation is very intuitive, and users can easily relate their hypothesis.
2. **Useful in Data exploration:** Decision tree is one of the fastest ways to identify the most significant variables and the relation between two or more variables. With the help of decision trees, we can create new variables/features that have a better power to predict the target variable. You can refer article (Trick to enhance the power of regression model) for one such trick. It can also be used in the data exploration stage. For example, we are working on a problem where we have information available in hundreds of variables; their decision tree will help to identify the most significant variable.
3. **Less data cleaning required:** It requires less data cleaning compared to some other modelling techniques. It is not influenced by outliers and missing values to a fair degree.
4. **Data type is not a constraint:** It can handle both numerical and categorical variables.
5. **Non-Parametric Method:** Decision tree is considered to be a non-parametric method. This means that decision trees have no assumptions about space distribution and the classifier structure.

Disadvantages

1. **Overfitting:** Overfitting is one of the most practical difficulties for decision tree models. This problem gets solved by setting constraints on model parameters and pruning (discussed in detail below).
2. **Not fit for continuous variables**: While working with continuous numerical variables, decision tree loses information when it categorizes variables in different categories.

Chapter 2: Regression Trees vs Classification Trees

We all know that the terminal nodes (or leaves) lie at the bottom of the decision tree. This means that decision trees are typically drawn upside down such that leaves are the bottom & roots are the tops (shown below).

Both the trees work almost similar to each other, let's look at the primary differences & similarity between classification and regression 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 the mean value.
3. In case of a classification tree, 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.
4. Both the trees divide the predictor space (independent variables) into distinct and non-overlapping regions. For the sake of simplicity, you can think of these regions as high dimensional boxes or boxes.

5. Both the trees follow a top-down greedy approach known as recursive binary splitting. We call it as ‘top-down’ because it begins from the top of the tree when all the observations are available in a single region and successively splits the predictor space into two new branches down the tree. It is known as ‘greedy’ because, the algorithm cares (looks for best variable available) about only the current split, and not about future splits which will lead to a better tree.
6. This splitting process is continued until a user-defined stopping criterion is reached. For example, we can tell the algorithm to stop once the number of observations per node becomes less than 50.
7. In both cases, the splitting process results in fully grown trees until the stopping criteria are reached. But, the fully grown tree is likely to overfit data, leading to poor accuracy on unseen data. This brings ‘pruning’. Pruning is one of the techniques used to tackle overfitting. We’ll learn more about it in the following section.

Chapter 3: How does a tree-based algorithm decide where to split?

The decision of making strategic splits heavily affects a tree's accuracy. The decision criteria are different for classification and regression trees.

Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that the purity of the node increases with respect to the target variable. Decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.

The algorithm selection is also based on the type of target variables. Let's look at the four most commonly used algorithms in the decision tree:

Gini

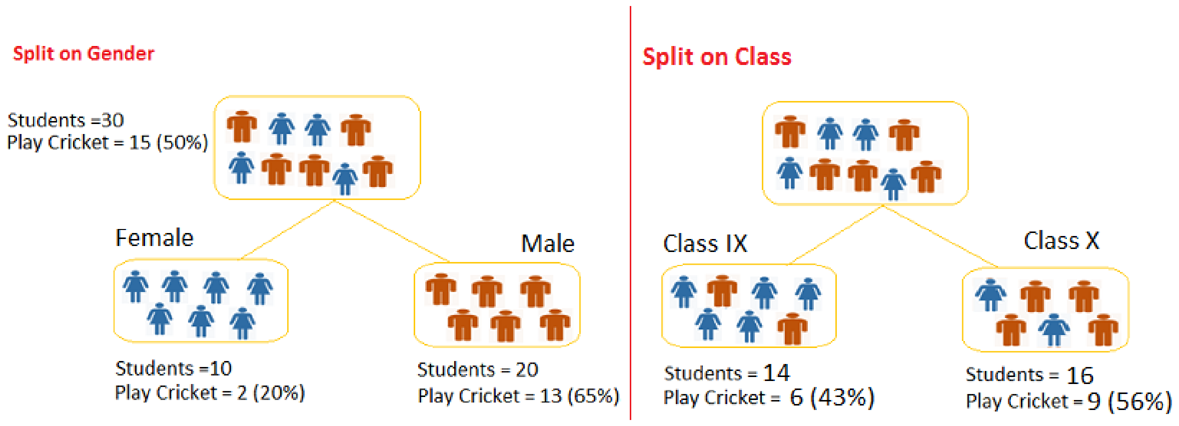
Gini says, if we select two items from a population at random, then they must be of the same class, and the probability for this is 1 if the population is pure.

1. It works with categorical target variable "Success" or "Failure".
2. It performs only Binary splits
3. Higher the value of Gini higher the homogeneity.
4. CART (Classification and Regression Tree) uses the Gini method to create binary splits.

Steps to Calculate Gini for a split

1. Calculate Gini for sub-nodes, using the formula sum of the square of probability for success and failure (p^2+q^2).
2. Calculate Gini for split using weighted Gini score of each node of that split

Example: – Referring to the example used above, where we want to segregate the students based on target variable (playing cricket or not). In the snapshot below, we split the population using two input variables Gender and Class. Now, I want to identify which split is producing more homogeneous sub-nodes using Gini.



Split on Gender:

1. Calculate, Gini for sub-node Female = (0.2)*(0.2)+(0.8)*(0.8)=0.68
2. Gini for sub-node Male = (0.65)*(0.65)+(0.35)*(0.35)=0.55
3. Calculate weighted Gini for Split Gender = (10/30)*0.68+(20/30)*0.55 = **0.59**

Similar for Split on Class:

1. Gini for sub-node Class IX = (0.43)*(0.43)+(0.57)*(0.57)=0.51
2. Gini for sub-node Class X = (0.56)*(0.56)+(0.44)*(0.44)=0.51
3. Calculate weighted Gini for Split Class = (14/30)*0.51+(16/30)*0.51 = **0.51**

Above, you can see that Gini score for *Split on Gender* is higher than *Split on Class,* hence, the node split will take place on Gender.

You might often come across the term ‘Gini Impurity’ which is determined by subtracting the Gini value from 1. So mathematically we can say,

Gini Impurity = 1-Gini

Chi-Square

It is an algorithm to find out the statistical significance between the differences between sub-nodes and parent node. We measure it by the sum of squares of standardized differences between observed and expected frequencies of the target variable.

1. It works with categorical target variable “Success” or “Failure”.
2. It can perform two or more splits.

3. Higher the value of Chi-Square higher the statistical significance of differences between sub-node and Parent node.
4. Chi-Square of each node is calculated using the formula
5. Chi-square = ((Actual – Expected) ^2 / Expected) ^1/2
6. It generates a tree called CHAID (Chi-square Automatic Interaction Detector)

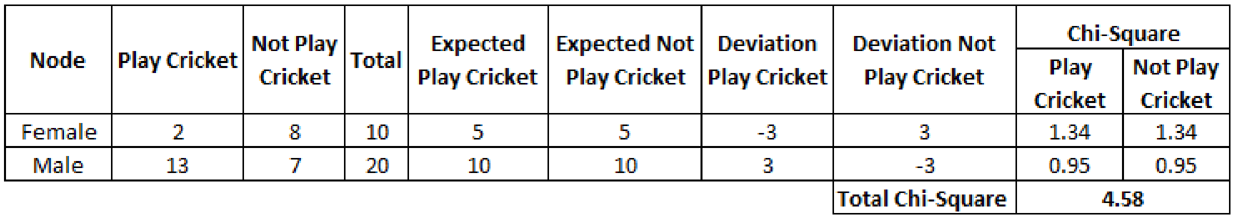
Steps to Calculate Chi-square for a split:

1. Calculate Chi-square for an individual node by calculating the deviation for Success and Failure both
2. Calculated Chi-square of Split using Sum of all Chi-square of success and Failure of each node of the split

Example: Let's work with the above example that we have used to calculate Gini.

Split on Gender:

1. First, we are populating for node Female, Populate the actual value for "**Play Cricket"** and **"Not Play Cricket"**, here these are 2 and 8 respectively.
2. Calculate expected value for "**Play Cricket"** and "**Not Play Cricket"**, here it would be 5 for both because the parent node has a probability of 50% and we have applied the same probability on Female count (10).
3. Calculate deviations by using the formula, Actual – Expected. It is for "**Play Cricket"** (2 – 5 = -3) and for "**Not play cricket"** (8 – 5 = 3).
4. Calculate Chi-square of nodes for "**Play Cricket**" and "**Not Play Cricket**" using the formula with formula, **= ((Actual – Expected) ^2 / Expected) ^1/2**. You can refer below table for calculation.
5. Follow similar steps for calculating Chi-square value for Male node.
6. Now add all Chi-square values to calculate Chi-square for split Gender.



Node	Play Cricket	Not Play Cricket	Total	Expected Play Cricket	Expected Not Play Cricket	Deviation Play Cricket	Deviation Not Play Cricket	Chi-Square	
								Play Cricket	Not Play Cricket
Female	2	8	10	5	5	-3	3	1.34	1.34
Male	13	7	20	10	10	3	-3	0.95	0.95
							Total Chi-Square	4.58	

Split on Class:

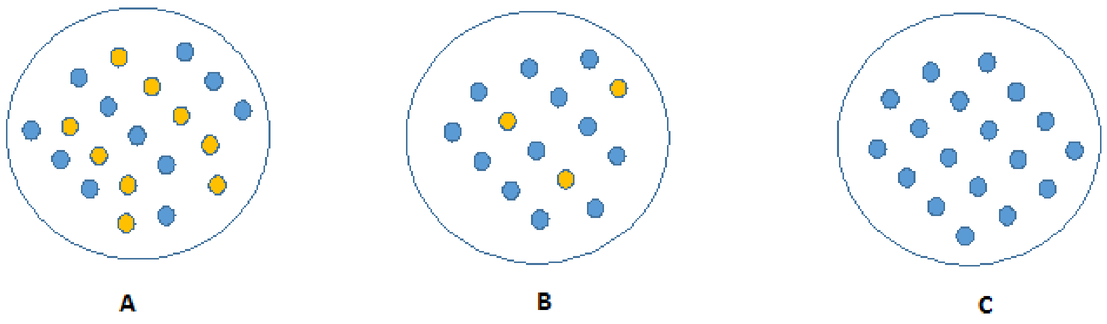
Perform similar steps of calculation for the split on Class, and you will come up with the below table.

Node	Play Cricket	Not Play Cricket	Total	Expected Play Cricket	Expected Not Play Cricket	Deviation Play Cricket	Deviation Not Play Cricket	Chi-Square	
								Play Cricket	Not Play Cricket
IX	6	8	14	7	7	-1	1	0.38	0.38
X	9	7	16	8	8	1	-1	0.35	0.35
							Total Chi-Square	1.46	

Above, you can see that Chi-square also identifies the Gender split is more significant compared to Class.

Information Gain

Look at the image below and think which nodes can be described easily. I am sure; your answer is C because it requires less information as all values are similar. On the other hand, B requires more information to describe it, and A requires the maximum information. In other words, we can say that C is a Pure node, B is less Impure, and A is more impure.



Now, we can build a conclusion that less impure node requires less information to describe it. And, the more impure node requires more information. Information theory is a measure to define this degree of disorganization in a system known as Entropy. If the sample is completely homogeneous, then the entropy is zero and if the sample is equally divided (50% – 50%), it has an entropy of one.

Entropy can be calculated using the formula:-

$$\text{Entropy} = -p \log_2 p - q \log_2 q$$

Here p and q are the probability of success and failure, respectively in that node. Entropy is also used with categorical target variables. It chooses the split which has the lowest entropy compared to the parent node and other splits. The lesser the entropy, the better it is.

Steps to calculate entropy for a split:

1. Calculate the entropy of the parent node
2. Calculate entropy of each individual node of split and calculate the weighted average of all sub-nodes available in the split.

Example: Let's use this method to identify the best split for student example.

1. Entropy for parent node = -(15/30) log2 (15/30) – (15/30) log2 (15/30) = **1**. Here 1 shows that it is an impure node.
2. Entropy for Female node = -(2/10) log2 (2/10) – (8/10) log2 (8/10) = 0.72 and for male node, -(13/20) log2 (13/20) – (7/20) log2 (7/20) = **0.93**
3. Entropy for split Gender = Weighted entropy of sub-nodes = (10/30) *0.72 + (20/30) *0.93 = **0.86**
4. Entropy for Class IX node, -(6/14) log2 (6/14) – (8/14) log2 (8/14) = 0.99 and for Class X node, -(9/16) log2 (9/16) – (7/16) log2 (7/16) = 0.99.
5. Entropy for split Class = (14/30) *0.99 + (16/30) *0.99 = **0.99**

Above, you can see that entropy for *Split on Gender* is the lowest among all, so the tree will split on *Gender*. We can derive information gain from entropy as **1- Entropy.**

Reduction in Variance

Till now, we have discussed the algorithms for the categorical target variable. Reduction in variance is an algorithm used for continuous target variables (regression problems). This algorithm uses the standard formula of variance to choose the best split. The split with lower variance is selected as the criteria to split the population:

$$\text{Variance} = \frac{\Sigma(X - \overline{X})^2}{n}$$

Above X-bar is mean of the values, X is actual, and n is the number of values.

Steps to calculate Variance:

1. Calculate variance for each node.
2. Calculate variance for each split as a weighted average of each node variance.

Example: - Let’s assign numerical value 1 for play cricket and 0 for not playing cricket. Now follow the steps to identify the right split:

1. Variance for Root node, here mean value is (15*1 + 15*0)/30 = 0.5 and we have 15 one and 15 zero. Now variance would be ((1-0.5) ^2+(1-0.5)^2+….15 times+(0-0.5)^2+(0-0.5)^2+…15 times) / 30, this can be written as (15*(1-0.5)^2+15*(0-0.5)^2) / 30 = **0.25**
2. Mean of Female node = (2*1+8*0)/10=0.2 and Variance = (2*(1-0.2)^2+8*(0-0.2)^2) / 10 = 0.16
3. Mean of Male Node = (13*1+7*0)/20=0.65 and Variance = (13*(1-0.65)^2+7*(0-0.65)^2) / 20 = 0.23
4. Variance for Split Gender = Weighted Variance of Sub-nodes = (10/30)*0.16 + (20/30) *0.23 = **0.21**
5. Mean of Class IX node = (6*1+8*0)/14=0.43 and Variance = (6*(1-0.43)^2+8*(0-0.43)^2) / 14= 0.24
6. Mean of Class X node = (9*1+7*0)/16=0.56 and Variance = (9*(1-0.56)^2+7*(0-0.56)^2) / 16 = 0.25
7. Variance for Split Gender = (14/30)*0.24 + (16/30) *0.25 = **0.25**

Above, you can see that Gender split has lower variance compared to the parent node, so the split would take place on the *Gender* variable.

Until here, we learnt about the basics of decision trees and the decision-making process involved to choose the best splits in building a tree model. As I said, the decision tree can be applied both on regression and classification problems. Let’s understand these aspects in detail.

Chapter 4: What are the key parameters of tree-based algorithms, and how can we avoid over-fitting in decision trees?

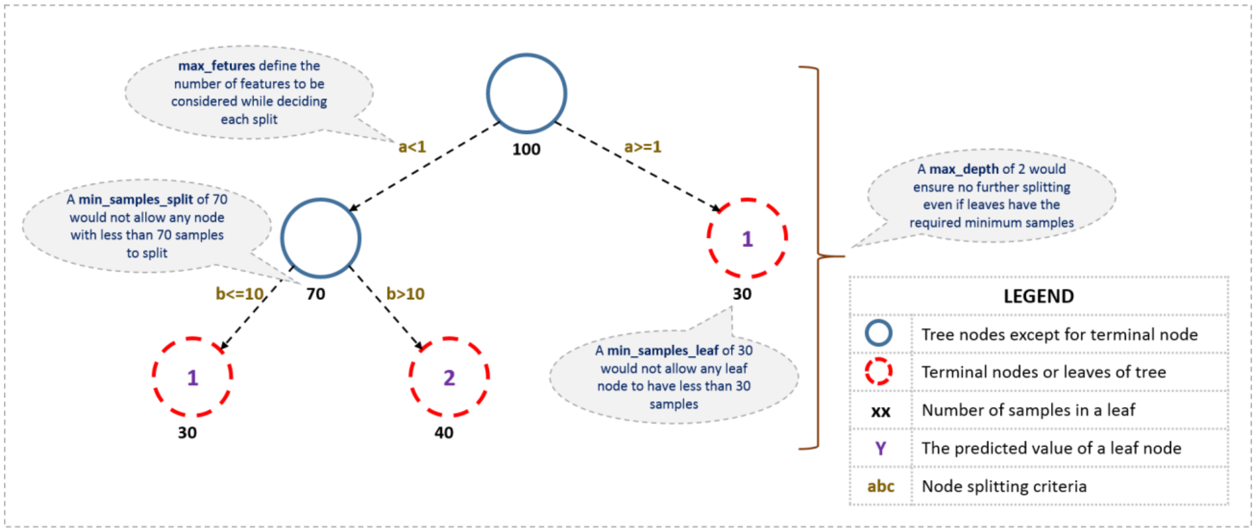
Overfitting is one of the key challenges faced while using tree-based algorithms. If there is no limit set of a decision tree, it will give you 100% accuracy on training set because in the worst case it will end up making 1 leaf for each observation. Thus, preventing overfitting is pivotal while modelling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

1. Setting constraints on tree size
2. Tree pruning

Let’s discuss both of these briefly.

Setting Constraints on tree-based algorithms

This can be done by using various parameters which are used to define a tree. First, let’s look at the general structure of a decision tree:


max_fetures define the number of features to be considered while deciding each split
a<1
100
a>=1
A min_samples_split of 70 would not allow any node with less than 70 samples to split
b<=10
70
b>10
1
30
2
40
1
30
A min_samples_leaf of 30 would not allow any leaf node to have less than 30 samples
A max_depth of 2 would ensure no further splitting even if leaves have the required minimum samples
LEGEND
Tree nodes except for terminal node
Terminal nodes or leaves of tree
xx Number of samples in a leaf
Y The predicted value of a leaf node
abc Node splitting criteria

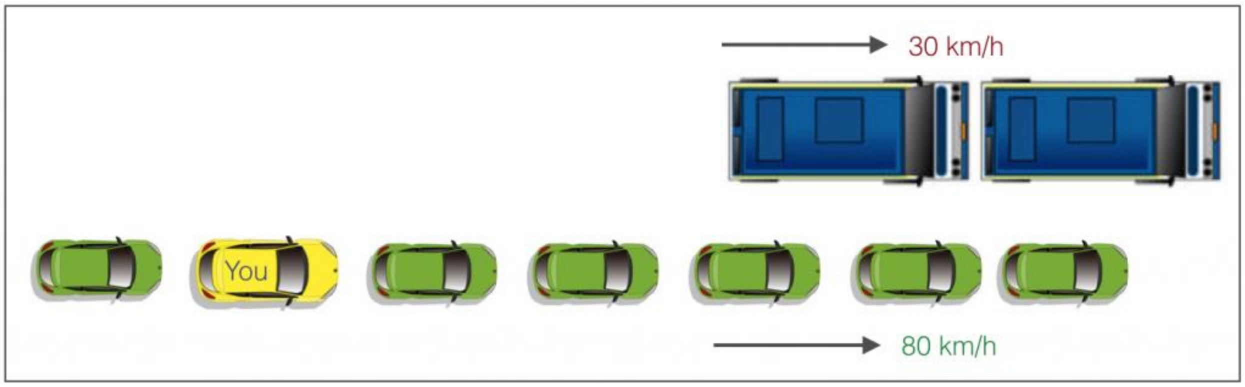
The parameters used for defining a tree are further explained below. The parameters described below are irrespective of the tool. It is important to understand the role of parameters used in tree modelling. These parameters are available in R & Python.

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 the majority will be very small.
3. **Maximum depth of the tree (vertical depth)**
    - The maximum depth of a tree.
    - Used to control over-fitting as higher depth will allow the 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 a split**
    - The number of features to consider while searching for the 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 up to 30-40% of the total number of features.
    - Higher values can lead to over-fitting, but it depends on case to case.

Pruning in tree-based algorithms

As discussed earlier, the technique of setting constraint is a greedy-approach. In other words, it will check for the best split instantaneously and move forward until one of the specified stopping conditions is reached. Let’s consider the following case when you’re driving:



There are 2 lanes:

1. A lane with cars moving at 80km/h
2. A lane with trucks moving at 30km/h

At this instant, you are the yellow car, and you have 2 choices:

1. Take a left and overtake the other 2 cars quickly
2. Keep moving in the present lane

Let’s analyze these choices. In the former choice, you’ll immediately overtake the car ahead and reach behind the truck and start moving at 30 km/h, looking for an opportunity to move back right. All cars originally behind you move ahead in the meanwhile. This would be the optimum choice if your objective is to maximize the distance covered in next say 10 seconds. In the latter choice, you sail through at same speed, cross trucks and then overtake maybe depending on the situation ahead. Greedy you!

This is exactly the difference between a normal decision tree & pruning. A decision tree with constraints won’t see the truck ahead and adopt a greedy approach by taking a left. On the other hand, if we use pruning, we, in effect, look at a few steps ahead and make a choice.

So, we know pruning is better. But how to implement it in a decision tree? The idea is simple.

1. We first make the decision tree to a large depth.
2. Then we start at the bottom and start removing leaves which are giving us negative returns when compared from the top.
3. Suppose a split is giving us a gain of say -10 (loss of 10) and then the next split on that gives us a gain of 20. A simple decision tree will stop at step 1, but in pruning, we will see that the overall gain is +10 and keep both leaves.

Note that sklearn's decision tree classifier does not currently support pruning. Advanced packages like XGBoost have adopted tree pruning in their implementation. But the library *rpart* in R provides a function to prune. Good for R users!

Chapter 5: Are tree-based algorithms better than linear models?

“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 variable is well approximated by a linear model, linear regression will outperform the tree-based model.
2. If there is a high non-linearity & 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!

Chapter 6: Working with tree-based algorithms Trees in R and Python

For R users and Python users, the decision tree is quite easy to implement. Let's quickly look at the set of codes that can get you started with this algorithm. For ease of use, I've shared standard codes where you'll need to replace your data set name and variables to get started.

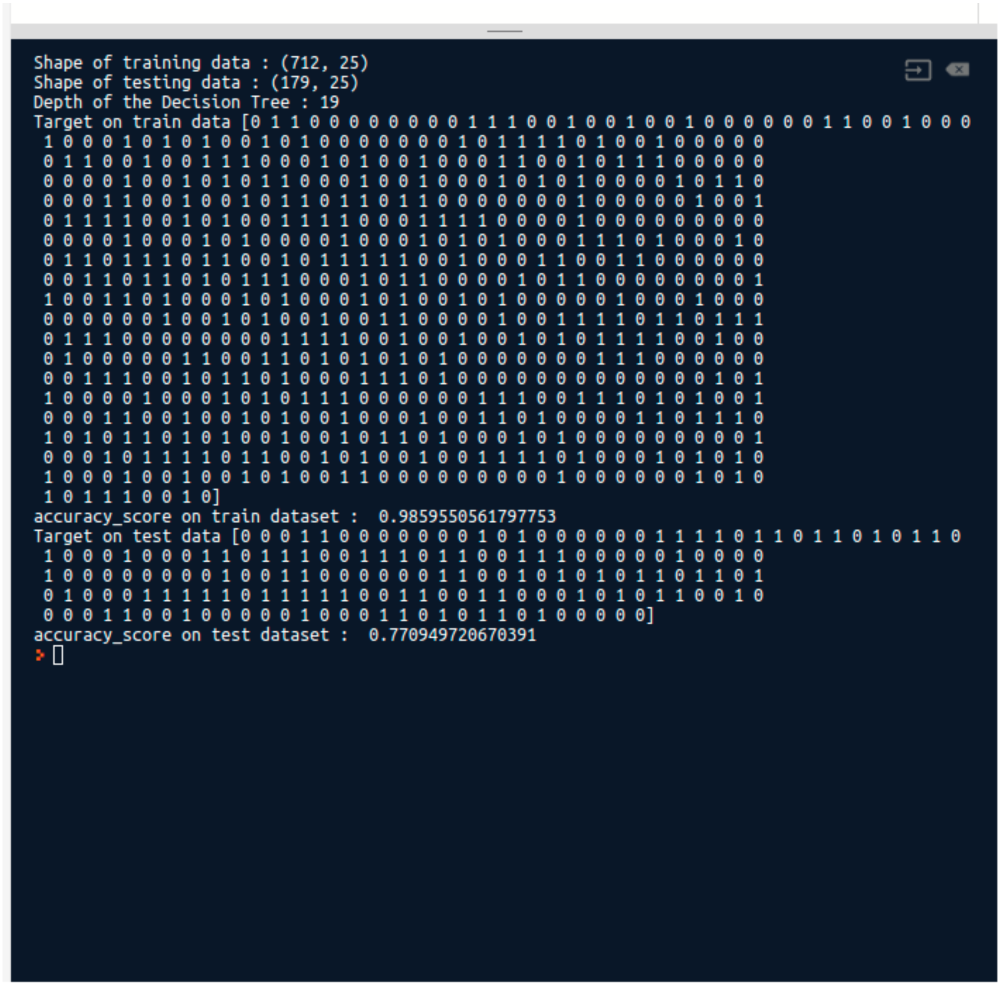
In fact, you can build the decision tree in Python right here! Here's a live coding window for you to play around the code and generate results:

Code

```
'''
The following code is for Decision Tree
Created by - Analytics Vidhya
'''


# importing required libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# read the train and test dataset
train_data = pd.read_csv('train-data.csv')
test_data = pd.read_csv('test-data.csv')

# shape of the dataset
print('Shape of training data :',train_data.shape)
print('Shape of testing data :',test_data.shape)

# Now, we need to predict the missing target variable in the test data
# target variable - Survived

# seperate the independent and target variable on training data
train_x = train_data.drop(columns=['Survived'],axis=1)
train_y = train_data['Survived']
```

```
# seperate the independent and target variable on testing data
test_x = test_data.drop(columns=['Survived'],axis=1)
test_y = test_data['Survived']

'''
Create the object of the Decision Tree model
You can also add other parameters and test your code here
Some parameters are : max_depth and max_features
Documentation of sklearn DecisionTreeClassifier:

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
'''
model = DecisionTreeClassifier()

# fit the model with the training data
model.fit(train_x,train_y)

# depth of the decision tree
print('Depth of the Decision Tree :', model.get_depth())

# predict the target on the train dataset
predict_train = model.predict(train_x)
print('Target on train data',predict_train)

# Accuray Score on train dataset
accuracy_train = accuracy_score(train_y,predict_train)
print('accuracy_score on train dataset : ', accuracy_train)

# predict the target on the test dataset
predict_test = model.predict(test_x)
print('Target on test data',predict_test)

# Accuracy Score on test dataset
accuracy_test = accuracy_score(test_y,predict_test)
print('accuracy_score on test dataset : ', accuracy_test)
```
Code Ends—

Output window



```
Shape of training data : (712, 25)
Shape of testing data : (179, 25)
Depth of the Decision Tree : 19
Target on train data [0 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 1 1 0 0 1 0 0 0
 1 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 1 1 1 0 1 0 0 1 0 0 0 0 0
 0 1 1 0 0 1 0 0 1 1 1 0 0 0 1 0 1 0 0 1 0 0 0 1 1 0 0 1 0 1 1 1 0 0 0 0 0
 0 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 1 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0 1 0 1 1 0
 0 0 0 1 1 0 0 1 0 0 1 0 1 1 0 1 1 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1
 0 1 1 1 1 0 0 1 0 1 0 0 1 1 1 1 0 0 0 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 1 1 1 0 1 0 0 0 1 0
 0 1 1 0 1 1 1 0 1 1 0 0 1 0 1 1 1 1 1 0 0 1 0 0 0 1 1 0 0 1 1 0 0 0 0 0 0
 0 0 1 1 0 1 1 0 1 0 1 1 1 0 0 0 1 0 1 1 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 1
 1 0 0 1 1 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0
 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 1 0 0 0 0 1 0 0 1 1 1 1 0 1 1 0 1 1 1
 0 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 1 0 0 1 0 0 1 0 1 0 1 1 1 1 0 0 1 0 0
 0 1 0 0 0 0 0 1 1 0 0 1 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0
 0 0 1 1 1 0 0 1 0 1 1 0 1 0 0 0 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 1 0 0 0 0 0 0 1 1 1 0 0 1 1 1 0 1 0 1 0 0 1
 0 0 0 1 1 0 0 1 0 0 1 0 1 0 0 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 1 1 0 1 1 1 0
 1 0 1 0 1 1 0 1 0 1 0 0 1 0 0 1 0 1 1 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1
 0 0 0 1 0 1 1 1 1 0 1 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1 0 1 0 0 0 1 0 1 0 1 0
 1 0 0 0 1 0 0 1 0 0 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0
 1 0 1 1 1 0 0 1 0]
accuracy_score on train dataset :  0.9859550561797753
Target on test data [0 0 0 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 1 1 1 0 1 1 0 1 1 0 1 0 1 1 0
 1 0 0 0 1 0 0 0 1 1 0 1 1 1 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 0 0 0 1 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 1 0 0 1 0 1 0 1 0 1 1 0 1 1 0 1
 0 1 0 0 0 1 1 1 1 1 0 1 1 1 1 1 0 0 1 1 0 0 1 1 0 0 0 1 0 1 0 1 1 0 0 1 0
 0 0 0 1 1 0 0 1 0 0 0 0 0 1 0 0 0 1 1 0 1 0 1 1 0 1 0 0 0 0 0]
accuracy_score on test dataset :  0.770949720670391
> ▯
```

For R users, there are multiple packages available to implement decision tree such as ctree, rpart, tree etc.

```
> library(rpart)

> x <- cbind(x_train,y_train)

# grow tree

> fit <- rpart(y_train ~ ., data = x,method="class")

> summary(fit)

#Predict Output

> predicted= predict(fit,x_test)
```

In the code above:

- y_train – represents dependent variable.
- x_train – represents independent variable
- x – represents training data.

For Python users, below is the code:

```
#Import Library
#Import other necessary libraries like pandas, numpy...
from sklearn import tree
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dataset
# Create tree object
model = tree.DecisionTreeClassifier(criterion='gini') # for classification, here you can change the algorithm as gini
or entropy (information gain) by default it is gini
# model = tree.DecisionTreeRegressor() for regression
# Train the model using the training sets and check score
model.fit(X, y)
model.score(X, y)
#Predict Output
predicted= model.predict(x_test)
```

Chapter 7: What are ensemble methods in tree-based algorithms

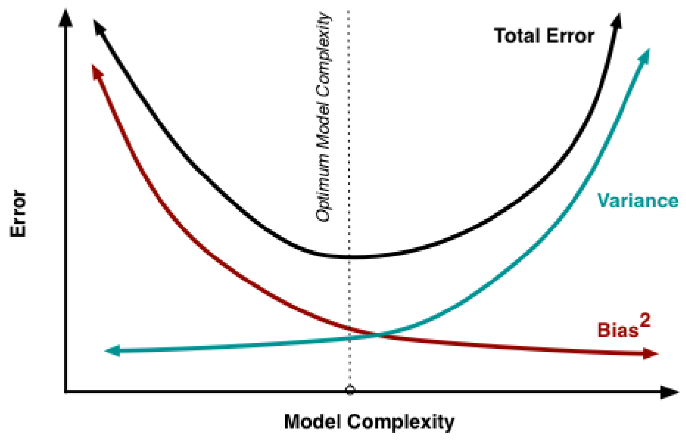
The literary meaning of the word ‘ensemble’ is a *group*. Ensemble methods involve a group of predictive models to achieve better accuracy and model stability. Ensemble methods are known to impart a supreme boost to tree-based models.

Like every other model, a tree-based algorithm 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 over-fitting your model, and your model will start suffering from high variance.

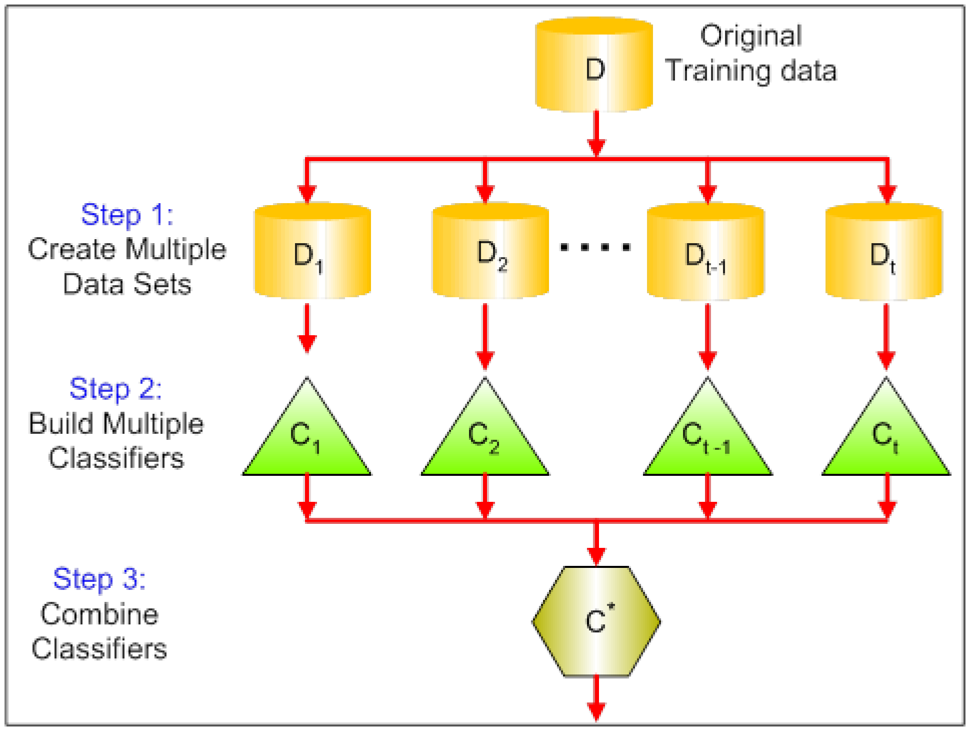
A champion model should maintain a balance between these two types of errors. This is known as the **trade-off management** of bias-variance errors. Ensemble learning is one way to execute this trade-off analysis.



Some of the commonly used ensemble methods include: Bagging, Boosting and Stacking. In this book, we’ll focus on Bagging and Boosting in detail.

Chapter 8: 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 modelled 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 DataSets**:
    - Sampling is done *with a 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 help in making robust models, less prone to overfitting
2. **Build Multiple Classifiers:**
    - Classifiers are built on each data set.
    - Generally, the same classifier is modelled 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.

Note that, here the number of models built is not a hyper-parameters. A higher number of models are always better or may give similar performance than lower numbers. It can be theoretically shown that the variance of the combined predictions is reduced to 1/n (n: number of classifiers) of the original variance, under some assumptions.

There are various implementations of bagging models. Random forest is one of them, and we'll discuss it next.

Chapter 9: What is Random Forest?

Random Forest is considered to be a *panacea* of all data science problems. On a funny note, when you can't think of any algorithm (irrespective of the situation), use random forest!

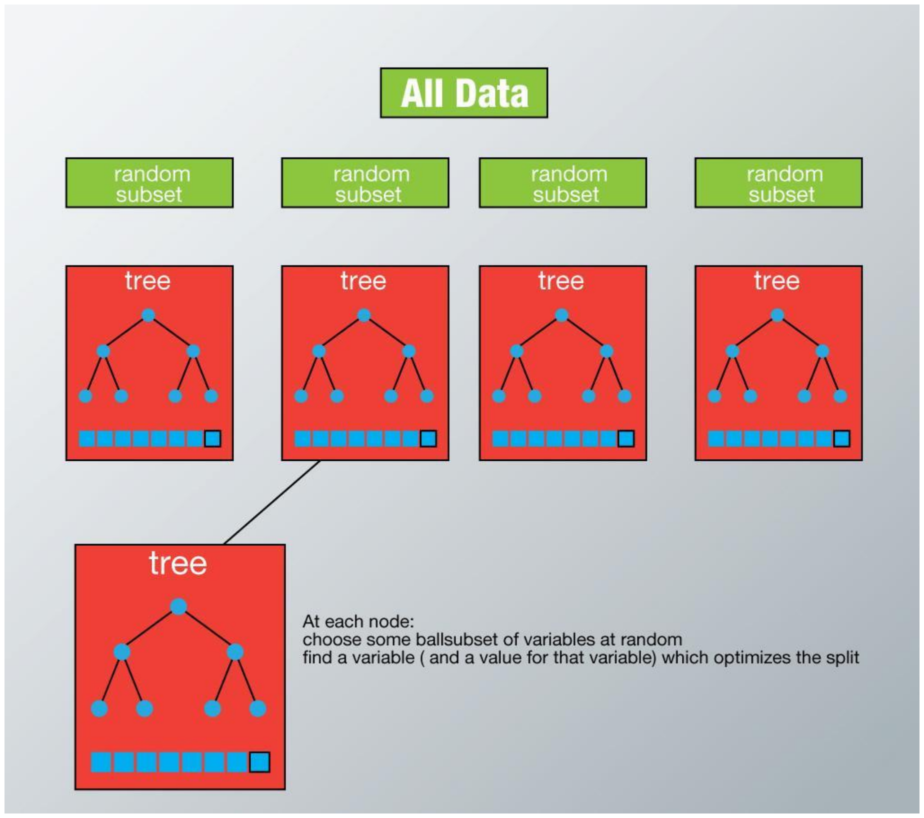
Random Forest is a versatile machine learning method capable of performing both regression and classification tasks. It also undertakes dimensional reduction methods, treats missing values, outlier values and other essential steps of data exploration, and does a fairly good job. It is a type of ensemble learning method, where a group of weak models combine to form a powerful model.

How does it work?

In Random Forest, we grow multiple trees as opposed to a single tree in the CART model (see the comparison between CART and Random Forest here, part1 and part2). To classify a new object based on attributes, each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest), and in case of regression, it takes the average of outputs by different trees.

It works in the following manner. Each tree is planted & grown as follows:

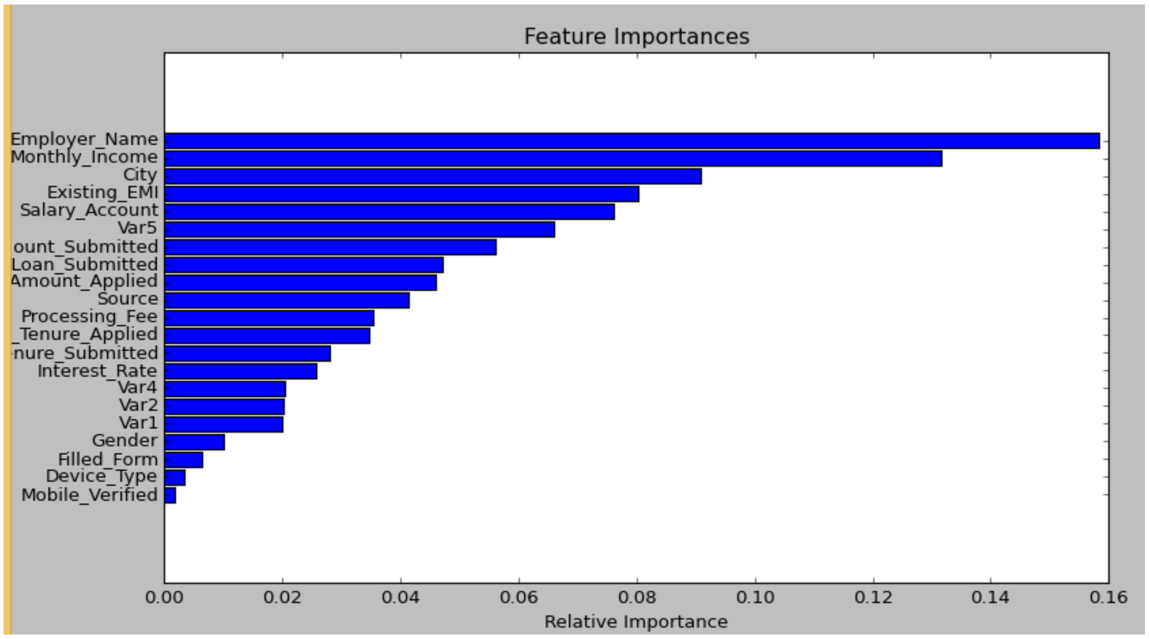
1. Assume the number of cases in the training set is N. Then, the sample of these N cases is taken at random but *with replacement*. This sample will be the training set for growing the tree.
2. If there are M input variables, a number m<M is specified such that at each node, m variables are selected at random out of the M. The best split on this m is used to split the node. The value of m is held constant while we grow the forest.
3. Each tree is grown to the largest extent possible, and there is no pruning.
4. Predict new data by aggregating the predictions of the n tree trees (i.e., majority votes for classification, the average for regression).


All Data
random subset
random subset
random subset
random subset
tree
tree
tree
tree
tree
At each node:
choose some ballsubset of variables at random
find a variable (and a value for that variable) which optimizes the split

To understand more in detail about this algorithm using a case study, please read this article "Introduction to Random forest – Simplified".

Advantages of Random Forest

- This algorithm can solve both types of problems, i.e. classification and regression and does a decent estimation on both fronts.
- One of the benefits of Random forest, which excites me most is the power of handling large data sets with higher dimensionality. It can handle thousands of input variables and identify the most significant variables, so it is considered as one of the dimensionality reduction methods. Further, the model outputs the **Importance of variable,** which can be a very handy feature (on some random data set).



- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.
- It has methods for balancing errors in data sets where classes are imbalanced.
- The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection.
- Random Forest involves the sampling of the input data with a replacement called bootstrap sampling. Here one-third of the data is not used for training and can be used for testing. These are called the **out of bag** samples. Error estimated on these out of bag samples is known as *out of bag error*. Study of error estimates by Out of the bag gives evidence to show that the out-of-bag estimate is as accurate as using a test set of the same size as the training set. Therefore, using the out-of-bag error estimate removes the need for a set-aside test set.

Disadvantages of Random Forest

- It surely does a good job at classification but not as good as for regression problem as it does not give precise continuous nature predictions. In case of regression, it doesn't predict beyond the range in the training data, and that they may over-fit data sets that are particularly noisy.
- Random Forest can feel like a black box approach for statistical modellers – you have very little control over what the model does. You can, at best – try different parameters and random seeds!

Python & R implementation

Random forests have commonly known implementations in R packages and Python scikit-learn. Let's look at the code of loading random forest model in R and Python below:

Code Starts here

```
'''
The following code is for the Random Forest
Created by - ANALYTICS VIDHYA
'''

# importing required libraries
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# read the train and test dataset
train_data = pd.read_csv('train-data.csv')
test_data = pd.read_csv('test-data.csv')

# view the top 3 rows of the dataset
print(train_data.head(3))

# shape of the dataset
print('\nShape of training data :',train_data.shape)
print('\nShape of testing data :',test_data.shape)
```

```
# Now, we need to predict the missing target variable in the test data
# target variable - Survived

# seperate the independent and target variable on training data
train_x = train_data.drop(columns=['Survived'],axis=1)
train_y = train_data['Survived']

# seperate the independent and target variable on testing data
test_x = test_data.drop(columns=['Survived'],axis=1)
test_y = test_data['Survived']

'''

Create the object of the Random Forest model
You can also add other parameters and test your code here
Some parameters are: n_estimators and max_depth
Documentation of sklearn RandomForestClassifier:

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

'''
model = RandomForestClassifier()

# fit the model with the training data
model.fit(train_x,train_y)

# number of trees used
print('Number of Trees used : ', model.n_estimators)

# predict the target on the train dataset
predict_train = model.predict(train_x)
print('\nTarget on train data',predict_train)

# Accuray Score on train dataset
accuracy_train = accuracy_score(train_y,predict_train)
```

```
print('\naccuracy_score on train dataset : ', accuracy_train)

# predict the target on the test dataset
predict_test = model.predict(test_x)
print('\nTarget on test data',predict_test)

# Accuracy Score on test dataset
accuracy_test = accuracy_score(test_y,predict_test)
print('\naccuracy_score on test dataset : ', accuracy_test)
```

Code ends here..

R Code

```
> library(randomForest)
> x <- cbind(x_train,y_train)
# Fitting model
> fit <- randomForest(Species ~ ., x,ntree=500)
> summary(fit)
#Predict Output
> predicted= predict(fit,x_test)
```

Chapter 10: What is Boosting?

Definition: The term ‘Boosting’ refers to a family of algorithms which converts weak learners to strong learners.

Let’s understand this definition in detail by solving a problem of spam email identification:

How would you classify an email as SPAM or not? Like everyone else, our initial approach would be to identify ‘spam’ and ‘not spam’ emails using the following criteria. If:

1. The email has only one image file (promotional image), It’s a SPAM
2. The email has only link(s), It’s a SPAM
3. Email body consists of a sentence like “You won prize money of $ xxxxxx”, It’s a SPAM
4. Email from our official domain “Analyticsvidhya.com”, Not a SPAM
5. Email from a known source, Not a SPAM

Above, we’ve defined multiple rules to classify an email into ‘spam’ or ‘not spam’. But, do you think these rules individually are strong enough to successfully classify an email? No.

Individually, these rules are not powerful enough to classify an email into ‘spam’ or ‘not spam’. Therefore, these rules are called **weak learners.**

To convert weak learner to strong learner, we’ll combine the prediction of each weak learner using methods like:

- Using average/ weighted average
- Considering prediction has a higher vote

For example: Above, we have defined 5 weak learners. Out of these 5, 3 are voted as ‘SPAM’, and 2 are voted as ‘Not a SPAM’. In this case, by default, we’ll consider an email as SPAM because we have a higher(3) vote for ‘SPAM’.

How does it work?

Now we know that boosting combines weak learner a.k.a. base learner to form a strong rule. An immediate question which should pop in your mind is, '*How boosting identifies weak rules?* '

To find a weak rule, we apply base learning (ML) algorithms with a different distribution. Each time base learning algorithm is applied, it generates a new weak prediction rule. This is an iterative process. After many iterations, the boosting algorithm combines these weak rules into a single strong prediction rule.

Here's another question which might haunt you, '*How do we choose different distributions for each round?'*

For choosing the right distribution, here are the following steps:

Step 1: The base learner takes all the distributions and assigns equal weight or attention to each observation.

Step 2: If there is any prediction error caused by the first base learning algorithm, then we pay greater attention to observations having prediction error. Then, we apply the next base learning algorithm.

Step 3: Iterate Step 2 till the limit of the base learning algorithm is reached or higher accuracy is achieved.

Finally, it combines the outputs from the weak learner and creates a strong learner which eventually improves the prediction power of the model. Boosting pays a higher focus on examples which are mis-classified or have higher errors by preceding weak rules.

There are many boosting algorithms which impart additional boost to a model's accuracy. In this book, we'll learn about the two most commonly used algorithms, i.e. Gradient Boosting (GBM) and XGBoost.

Chapter 11: Which is more powerful: GBM or XGBoost?

I've always admired the boosting capabilities of the XGBoost algorithm. At times, I've found that it provides better results compared to GBM implementation, but at times you might find that the gains are just marginal. When I explored more about its performance and science behind its high accuracy, I discovered many advantages of XGBoost over GBM:

1. **Regularization:**
    - Standard GBM implementation has no regularization like XGBoost; therefore, it also helps to reduce overfitting.
    - In fact, XGBoost is also known as '**regularized boosting**' technique.
2. **Parallel Processing:**
    - XGBoost implements parallel processing and is **blazingly faster** as compared to GBM.
    - But hang on, we know that boosting is a sequential process, so how can it be parallelized? We know that each tree can be built only after the previous one, so what stops us from making a tree using all cores? I hope you get where I'm coming from. Check this link out to explore further.
    - XGBoost also supports implementation on Hadoop.
3. **High Flexibility**
    - XGBoost allows users to define **custom optimization objectives and evaluation criteria**.
    - This adds a whole new dimension to the model, and there is no limit to what we can do.
4. **Handling Missing Values**
    - XGBoost has an in-built routine to handle missing values.
    - Users are required to supply a different value than other observations and pass that as a parameter. XGBoost tries different things as it encounters a missing value on each node and learns which path to take for missing values in future.
5. **Tree Pruning:**
    - A GBM would stop splitting a node when it encounters a negative loss in the split. Thus, it is more of a **greedy algorithm**.
    - XGBoost, on the other hand, makes **splits up to the max_depth** specified and then starts **pruning** the tree backwards and removing splits beyond which there is no positive gain.
    - Another advantage is that sometimes a split of negative loss say -2 may be followed by a split of positive loss +10. GBM would stop as it encounters -2. But XGBoost will go deeper, and it will see a combined effect of +8 of the split and keep both.

6. **Built-in Cross-Validation**
    - XGBoost allows users to run **cross-validation at each iteration** of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run.
    - This is unlike GBM, where we have to run a grid-search, and only limited values can be tested.
7. **Continue on Existing Model**
    - Users can start training an XGBoost model from its last iteration of the previous run. This can be of significant advantage in certain specific applications.
    - GBM implementation of sklearn also has this feature, so they are even on this point.

Chapter12: Working with GBM in R and Python

Before we start working, let’s quickly understand the important parameters and the working of this algorithm. This will be helpful for both R and Python users. Below is the overall pseudo-code of GBM algorithm for 2 classes:

```
1. Initialize the outcome

2. Iterate from 1 to the total number of trees

  2.1 Update the weights for targets based on the previous run (higher for the ones misclassified)

  2.2 Fit the model on a selected subsample of data

  2.3 Make predictions on the full set of observations

  2.4 Update the output with current results taking into account the learning rate

3. Return the final output.
```

This is an extremely simplified (probably naive) explanation of GBM’s working. But, it will help every beginner to understand this algorithm.

Let's consider the important GBM parameters used to improve model performance in Python:

1. **learning_rate**
    - This determines the impact of each tree on the final outcome (step 2.4). GBM works by starting with an initial estimate which is updated using the output of each tree. The learning parameter controls the magnitude of this change in the estimates.
    - Lower values are generally preferred as they make the model robust to the specific characteristics of a tree and thus allowing it to generalize well.
    - Lower values would require a higher number of trees to model all the relations and will be computationally expensive.
2. **n_estimators**
    - The number of sequential trees to be modelled (step 2)
    - Though GBM is fairly robust at a higher number of trees, it can still overfit at a point. Hence, this should be tuned using CV for a particular learning rate.
3. **subsample**
    - The fraction of observations to be selected for each tree. Selection is made by random sampling.

    - Values slightly less than 1 make the model robust by reducing the variance.
    - Typical values ~0.8 generally work fine but can be fine-tuned further.

Apart from these, there are certain miscellaneous parameters which affect overall functionality:

1. **loss**
    - It refers to the loss function to be minimized in each split.
    - It can have various values for classification and regression case. Generally, the default values work fine. Other values should be chosen only if you understand their impact on the model.
2. **init**
    - This affects the initialization of the output.
    - This can be used if we have made another model whose outcome is to be used as the initial estimates for GBM.
3. **random_state**
    - The random number seed, so that same random numbers are generated every time.
    - This is important for parameter tuning. If we don't fix the random number, then we'll have different outcomes for subsequent runs on the same parameters, and it becomes difficult to compare models.
    - It can potentially result in overfitting to a particular random sample selected. We can try running models for different random samples, which is computationally expensive and generally not used.
4. **verbose**
    - The type of output to be printed when the model fits. The different values can be:
        - 0: no output generated (default)
        - 1: output generated for trees at certain intervals
        - >1: output generated for all trees
5. **warm_start**
    - This parameter has an interesting application and can help a lot if used judicially.
    - Using this, we can fit additional trees on previous fits of a model. It can save a lot of time, and you should explore this option for advanced applications
6. **presort**
    - Select whether to presort data for faster splits.
    - It makes the selection automatically by default, but it can be changed if needed.

I know it's a long list of parameters, but I have simplified it for you in an excel file which you can download from this GitHub repository.

For R users, using the caret package, there are 3 main tuning parameters:

1. *n.trees* – It refers to the number of iterations i.e. tree which will be taken to grow the trees
2. *interaction.depth* – It determines the complexity of the tree i.e. total number of splits it has to perform on a tree (starting from a single node)
3. *shrinkage* – It refers to the learning rate. This is similar to learning_rate in python (shown above).
4. n.minobsinnode – It refers to the minimum number of training samples required in a node to perform splitting

GBM in R (with cross validation)

I've shared the standard codes in R and Python. At your end, you'll be required to change the value of dependent variable and data set name used in the codes below. Considering the ease of implementing GBM in R, one can easily perform tasks like cross validation and grid search with this package.

```
> library(caret)

> fitControl <- trainControl(method = "cv",

                   number = 10, #5folds)

> tune_Grid <-  expand.grid(interaction.depth = 2,

                    n.trees = 500,

                    shrinkage = 0.1,

                    n.minobsinnode = 10)

> set.seed(825)

> fit <- train(y_train ~ ., data = train,

            method = "gbm",

            trControl = fitControl,

            verbose = FALSE,

            tuneGrid = gbmGrid)

> predicted= predict(fit,test,type= "prob")[,2]
```

GBM in Python

```
'''
The following code is for Gradient Boosting
Created by - ANALYTICS VIDHYA
'''

# importing required libraries
import pandas as pd
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score

# read the train and test dataset
train_data = pd.read_csv('train-data.csv')
test_data = pd.read_csv('test-data.csv')

# shape of the dataset
print('Shape of training data :',train_data.shape)
print('Shape of testing data :',test_data.shape)

# Now, we need to predict the missing target variable in the test data
# target variable - Survived

# seperate the independent and target variable on training data
train_x = train_data.drop(columns=['Survived'],axis=1)
train_y = train_data['Survived']

# seperate the independent and target variable on testing data
test_x = test_data.drop(columns=['Survived'],axis=1)
test_y = test_data['Survived']
```

```
'''
Create the object of the GradientBoosting Classifier model
You can also add other parameters and test your code here
Some parameters are : learning_rate, n_estimators
Documentation of sklearn GradientBoosting Classifier:

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
'''
model = GradientBoostingClassifier(n_estimators=100,max_depth=5)

# fit the model with the training data
model.fit(train_x,train_y)

# predict the target on the train dataset
predict_train = model.predict(train_x)
print('\nTarget on train data',predict_train)

# Accuray Score on train dataset
accuracy_train = accuracy_score(train_y,predict_train)
print('\naccuracy_score on train dataset : ', accuracy_train)

# predict the target on the test dataset
predict_test = model.predict(test_x)
print('\nTarget on test data',predict_test)

# Accuracy Score on test dataset
accuracy_test = accuracy_score(test_y,predict_test)
print('\naccuracy_score on test dataset : ', accuracy_test
```

Code ends here.

Working with XGBoost in R and Python

XGBoost (eXtreme Gradient Boosting) is an advanced implementation of gradient boosting algorithm. It's a feature to implement parallel computing makes it at least **10 times faster** than existing gradient boosting implementations. It supports various objective functions, including regression, classification and ranking.

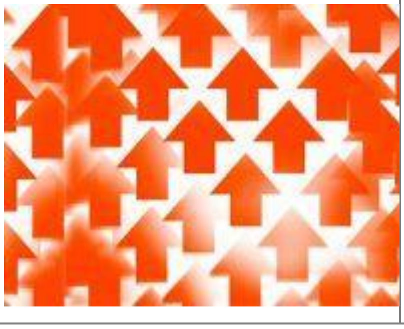
R Tutorial: For R users, this is a complete tutorial on XGBoost which explains the parameters along with codes in R. Check Tutorial.

Python Tutorial: For Python users, this is a comprehensive tutorial on XGBoost, good to get you started. Check Tutorial.

Chapter 13: Where to practice?

The practice is the one and true method of mastering any concept. Hence, you need to *start practising* if you wish to master these algorithms.

Till here, you've gained significant knowledge on tree-based algorithms along with these practical implementations. It’s time that you start working on them. Here are open practice problems where you can participate and check your live rankings on the leaderboard:

	Practice Problem: Food Demand Forecasting Challenge	Predict the demand of meals for a meal delivery company
	Practice Problem: HR Analytics Challenge	Identify the employees most likely to get promoted
	Practice Problem: Predict Number of Upvotes	Predict the number of upvotes on a query asked at an online question & answer platform

Endnotes

Tree-based algorithms are important for every data scientist to learn. In fact, tree models are known to provide the best model performance in the family of whole machine learning algorithms. In this book, we learned about GBM and XGBoost. And with this, we come to the end of this book.

We discussed tree-based algorithms from scratch. We learnt the importance of a decision tree and how that simple concept is being used in boosting algorithms. For better understanding, I would suggest you to continue practicing these algorithms practically. Also, do keep a note of the parameters associated with boosting algorithms. I'm hoping that this book would enrich you with complete knowledge of tree-based modelling.

Did you find this book useful? If you have experienced, what's the best trick you've used while using tree-based models? Feel free to share your tricks, suggestions and opinions in the comments section below.

Note – The discussions of this article are going on at AV's Discuss portal. Join here!

You can test your skills and knowledge. Check out Live Competitions and compete with best Data Scientists from all over the world.

You can also read this article on Analytics Vidhya's Android APP

Analytics Vidhya Community Discussion forums

Analytics Vidhya discussions is a question and answer site for data science professionals or people wanting to enter data science.

If you have any questions on analytics tools and techniques, visualizations, Big Data, machine learning - feel free to ask them here. Our community loves any question on this topic - be it a specific question like:

Question: While implementing this piece of code, I am getting following error?

or a generic, high-level question like:

Question: How do I become a data scientist?

So, start your journey by asking questions or sharing your perspective with other data science professionals. Click on the link given below and join the discussion.

https://discuss.analyticsvidhya.com/