

A Dive Into Decision Trees

How do Decision Trees work?

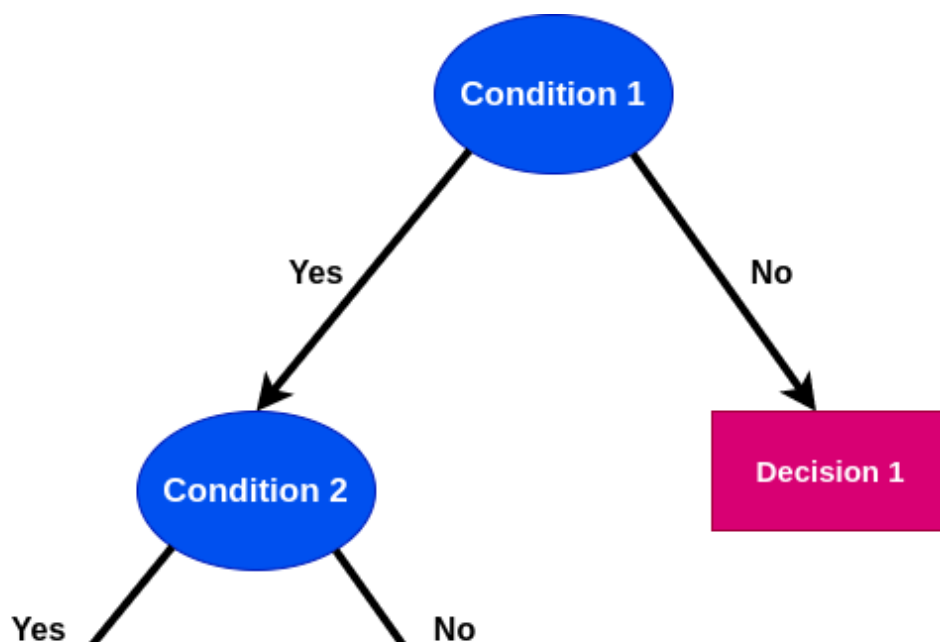


Abhijit Roy · Nov 6, 2020 · 12 min read

Decision Trees are some of the most used machine learning algorithms. They are used for both classification and Regression. They can be used for both linear and non-linear data, but they are mostly used for non-linear data. Decision Trees as the name suggests works on a set of decisions derived from the data and its behavior. It does not use a linear classifier or regressor, so its performance is independent of the linear nature of the data. Boosting and Bagging algorithms have been developed as ensemble models using the basic principle of decision trees compiled with some modifications to overcome some important drawbacks of decision trees and provide better results. One of the other most important reasons to use tree models is that they are very easy to interpret.

Decision Trees

Decision Trees can be used for both classification and regression. The methodologies are a bit different, though principles are the same. The decision trees use the CART algorithm (Classification and Regression Trees). In both cases, decisions are based on conditions on any of the features. The internal nodes represent the conditions and the leaf nodes represent the decision based on the conditions.





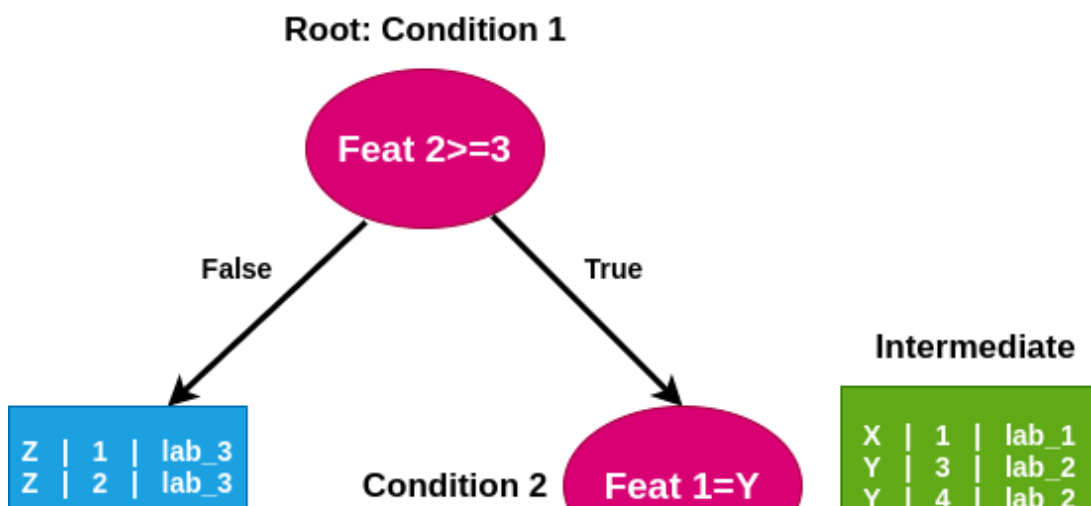
Classification

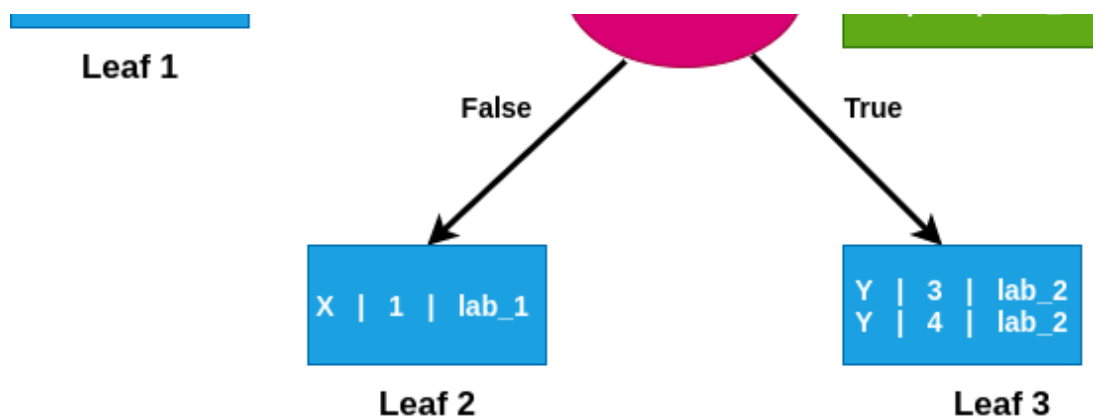
A decision tree is a graphical representation of all possible solutions to a decision based on certain conditions.

On each step or node of a decision tree, used for classification, we try to form a condition on the features to separate all the labels or classes contained in the dataset to the fullest purity. Let's see how the idea works.

Feature 1	Feature 2	Label
X	4	lab_1
Y	3	lab_2
Z	1	lab_3
Y	4	lab_2
Z	2	lab_3

Say, the table given above is our dataset. If we try to analyze the dataset and create a decision tree based on the dataset, we will obtain something like the tree given below





In the root node, we are using “Feat 2 \geq 3” as the condition to segregate our dataset for the first time. We can see if the answer is false, we reach a leaf, that has only the entries with label 3. So, label 3 is completely separated. But, if the answer is True, we reach an intermediate node. In this case, there are 3 entries, 2 of which belong to label 2 and one belongs to label 1. So, this result has impurity as there are two labels mixed. We apply another condition on the intermediate state and obtain the purely separated labels. On Leaf 2, we have obtained only the Label 1 entry and on leaf 3, we have only the Label 2 entries.

Now, the questions that may arise are:

1. How do we fix the deciding value in case of numerical features, for instance, “feat 2 \geq 3”?
2. How do we decide which features should we use to create our internal condition nodes?
3. How to decide which feature to use first, for instance, in the previous case, we had feature 1 and feature 2, so, which feature should we use as the root?

We will see the answers to these questions as we move forward.

Before we jump into the details of those answers let’s look at some definitions which will play major roles in answering the questions.

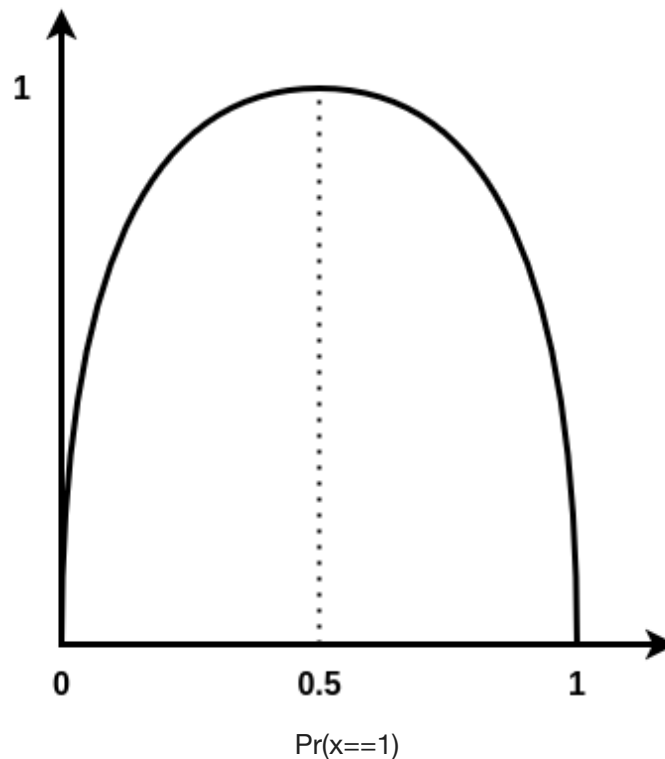
Entropy: It gives the measure of impurity or randomness in the data. It is given by:

$$Entropy = - P(class\ 1) \times \log(P(class\ 1)) - P(class\ 2) \times \log(P(class\ 2))$$

where P denotes the probability.

If there are two classes, class 1 and class 2, of equal numbers, i.e, the number of entries of class 1 is equal to the number of entries of class 2, and we randomly select an entry, it will belong to any class 1 or 2 with a 50% probability each. In such cases, the entropy will be high.

If a certain dataset has all data belonging to either class 1 or class 2, the entropy obtained is 0, as in that case $P(\text{class1})$ or $P(\text{class2})$ will be equal to 0. If $P(\text{class1})=0$ then $P(\text{class2})$ should be equal to 1. So, it is evident that the entropy will be high if we have impure or mixed class labels in a dataset.



The above diagram shows the variation of the probability of labels with entropy. We can see if the probability of a label is 0.5, the entropy is the maximum.

If in a dataset there are a total of 20 entries or rows, and 14 of them belongs to label 1 and 6 of them belongs to label 2, the entropy will be equal to:

$$= -P(\text{label } 1) \cdot \log(P(\text{label } 1)) - P(\text{label } 2) \cdot \log(P(\text{label } 2))$$

$$= -14/20 \cdot \log(14/20) - 6/20 \cdot \log(6/20)$$

$$=0.880$$

Information Gain: The information gain is the decrease in the entropy after the dataset is split on the basis of an attribute. Constructing a decision tree depends on finding the attribute that returns the highest information gain. It helps in choosing which feature or attribute will be used to create the deciding internal node at a particular point.

It is given by:

$$\text{Information gain} = \text{Entropy}(s) - [(\text{Weighted average}) \times (\text{Entropy of each feature})]$$

Now, how does this actually operate?

Feature 1	Feature 2	Label
1	1	lab_1
1	1	lab_1
2	1	lab_2
3	2	lab_2
3	3	lab_2
3	3	lab_1
2	3	lab_2
1	2	lab_1
1	3	lab_2
2	2	lab_2

Say, this is our dataset. So, we have feature 1 and feature 2, which should be our root node. The above decision is based on the amount of information gain each of the features provides. So, let's check.

Firstly, we want to check the distribution of the labels based on each feature, in order to get an insight into how much information gain will a feature provide.

For instance for feature 1,

Feature 1==1		Feature 1==2		Feature 1==3
labels		labels		labels
lab_1		lab_2		lab_2
lab_1		lab_2		lab_2
lab_1		lab_2		lab_1
lab_2				

We can see if we select the condition “Feature 1 == 2”, we can successfully separate label 2 from the dataset purely.

Now, Entropy ($E(s)$) = $-\frac{4}{10} \log(\frac{4}{10}) - \frac{6}{10} \log(\frac{6}{10}) = 0.97$

For Feature 1:

$E(\text{feature_1} == 1) = -\frac{3}{4} \log(\frac{3}{4}) - \frac{1}{4} \log(\frac{1}{4}) = 0.81$

$E(\text{feature_1} == 2) = -\frac{3}{3} \log(\frac{3}{3}) = -1 \log 1 = 0$

$E(\text{feature_1} == 3) = -\frac{2}{3} \log(\frac{2}{3}) - \frac{1}{3} \log(\frac{1}{3}) = 0.91$

For Feature 1, the weighted average =


$\frac{4}{10} * 0.81 + \frac{3}{10} * 0 + \frac{3}{10} * 0.91 = 0.597$

Information Gain = $0.97 - 0.597 = 0.313$

Again, let's do the same for feature 2:

For Feature 2:

Feature 2==1		Feature 2==2		Feature 2==3
labels		labels		labels
lab_1		lab_2		lab_2
lab_1		lab_1		lab_1

lab_2		lab_2		lab_2
				lab_2 

For Feature 2:

$$E(\text{feature_2}=1) = -\frac{2}{3} \log\left(\frac{2}{3}\right) - \frac{1}{3} \log\left(\frac{1}{3}\right) = 0.91$$

$$E(\text{feature_2}=2) = -\frac{2}{3} \log\left(\frac{2}{3}\right) - \frac{1}{3} \log\left(\frac{1}{3}\right) = 0.91$$

$$E(\text{feature_2}=3) = -\frac{3}{4} \log\left(\frac{3}{4}\right) - \frac{1}{4} \log\left(\frac{1}{4}\right) = 0.81$$

For Feature 2, the weighted average =

$$\frac{3}{10} \cdot 0.91 + \frac{3}{10} \cdot 0.91 + \frac{4}{10} \cdot 0.81 = 0.87$$

$$\text{So, Information Gain for feature 2} = 0.97 - 0.87 = 0.1$$

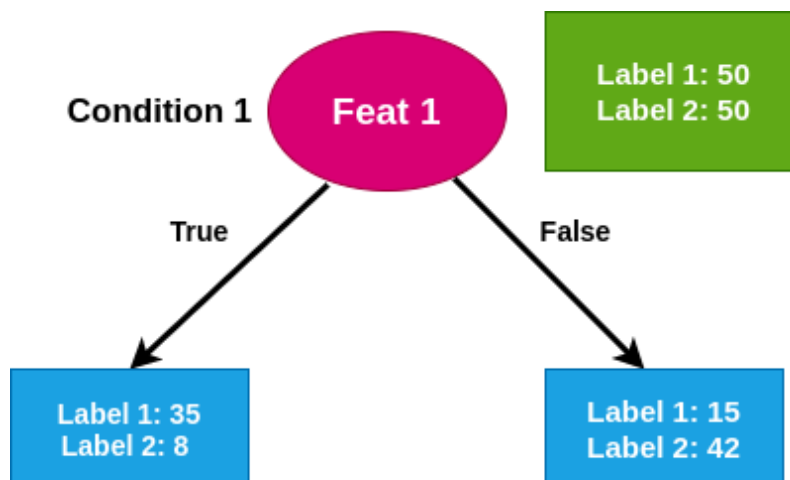
Now, we found information gain at the point for feature 1 is larger than that of feature 2. So, we will use feature 1 to form the root node.

Gini index: Gini Index is also a measure of impurity used to build a decision tree using the CART mechanism.

It is given by:

$$\text{Gini Impurity} = 1 - (\text{Probability of 'Class 1'})^2 - (\text{Probability of 'Class 2'})^2$$

Let's see its working also,



Say, this is our distribution on a dataset of 100 entries. A condition applied to Feature 1 gives the above conclusion.

Gini Impurity of Leaf 1: $1 - (35/43)^2 - (8/43)^2 = 0.302$

Gini Impurity of Leaf 2: $1 - (15/57)^2 - (42/57)^2 = 0.38$

Now, the overall Gini Impurity for Feature 1:

The weighted average of Gini impurity:

*$= 43/100 * 0.30 + 57/100 * 0.38 = 0.34$*

Similarly, we calculate the Gini impurity index for other features also.

One thing to notice is that if every entry in a dataset belongs to only 1 class, i.e, either class 1 or class 2,


Gini Impurity: $1 - (1/(0+1)^2) - (0/(0+1)^2) = 0$

So, we can see for a pure distribution the Gini impurity index is 0.

Gini Impurity index can also be used to decide which feature should be used to create the condition node. The feature that results in a smaller Gini impurity index is chosen to create the internal condition node at that point.

We have seen the concepts, we required to know in order to understand the working of the decision tree. I think we have found the answers to our second and third questions, i.e, how the features are decided and which feature is used to form the condition for the conclusion. We need to now find the answer to the first question, i.e, how to obtain the values which are used to form the condition in case of continuous numerical features.

The process to find the best suitable value to create a condition is pretty easy. First, we sort the dataset based on the numerical valued feature in an ascending manner. Next, we find the average of the adjacent pairs of numeric values.

Feature	Value averaged
Val_1	
	avg_1
Val_2	
	avg_2
Val_3	
	avg_3
Val_4	
	avg_4
Val_5	
	avg_5 
Val_6	

Say, we obtained the averages as shown above. So, $avg_1 = (val_1 + val_2)/2$. Next, we fit the average values in the condition to check which provides the minimum Gini Impurity index or which provides the maximum information gain, based on the methods we are using. The average value serves as the cutoff value in our condition formed for the feature.

Similarly, for discrete value-based or class-based features, we try and fit every value present in the set and create the condition. Finally, we select the value or condition that gives the minimum Gini impurity index. So, this wraps up our first query.

Now, we have seen we consider the Gini impurity index or Information gain to decide which condition we should consider. In some cases, some features show no improvement at all or 0 information gain, such features are never used in a decision tree. This is how the decision tree does automatic feature selection.

One of the chief challenges of the decision tree is that it results in overfitting the data. This is mostly because of the fact, that it creates a condition-based approach to the training data. So, it fits very tightly on the training data. Now, the more the conditions applied on the train data in the tree, the deeper it grows, the tighter it fits the data.

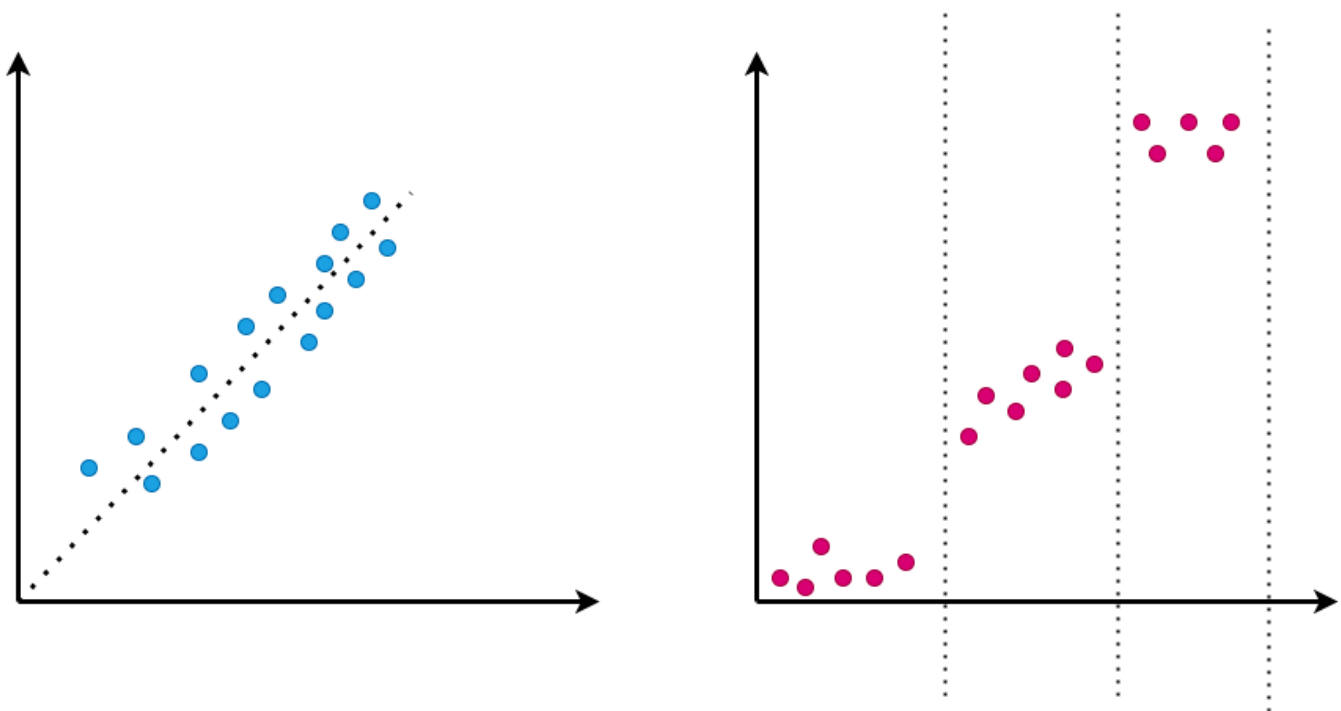
But after a point, it starts taking into consideration very small changes on some features, that give very low information gain, mostly these points destroy the generalization of the model and behave like outliers on the test data. We can restrict these conditions by using a threshold on the information gain, such that, if the information gain provided by the condition is less than a given value, we won't consider the condition. This partially prevents overfitting and helps create a generalized model.

Pruning is a method of removal of nodes to get the optimal solution and a tree with reduced complexity. It removes branches or nodes in order to create a sub-tree that has reduced overfitting tendency. We will talk about the concept once we are done with Regression trees.

Regression

To understand the concept of regression trees, we must have a clear idea about the concept of regression and linear regression. If needed, feel free to go through [my article on regression](#).

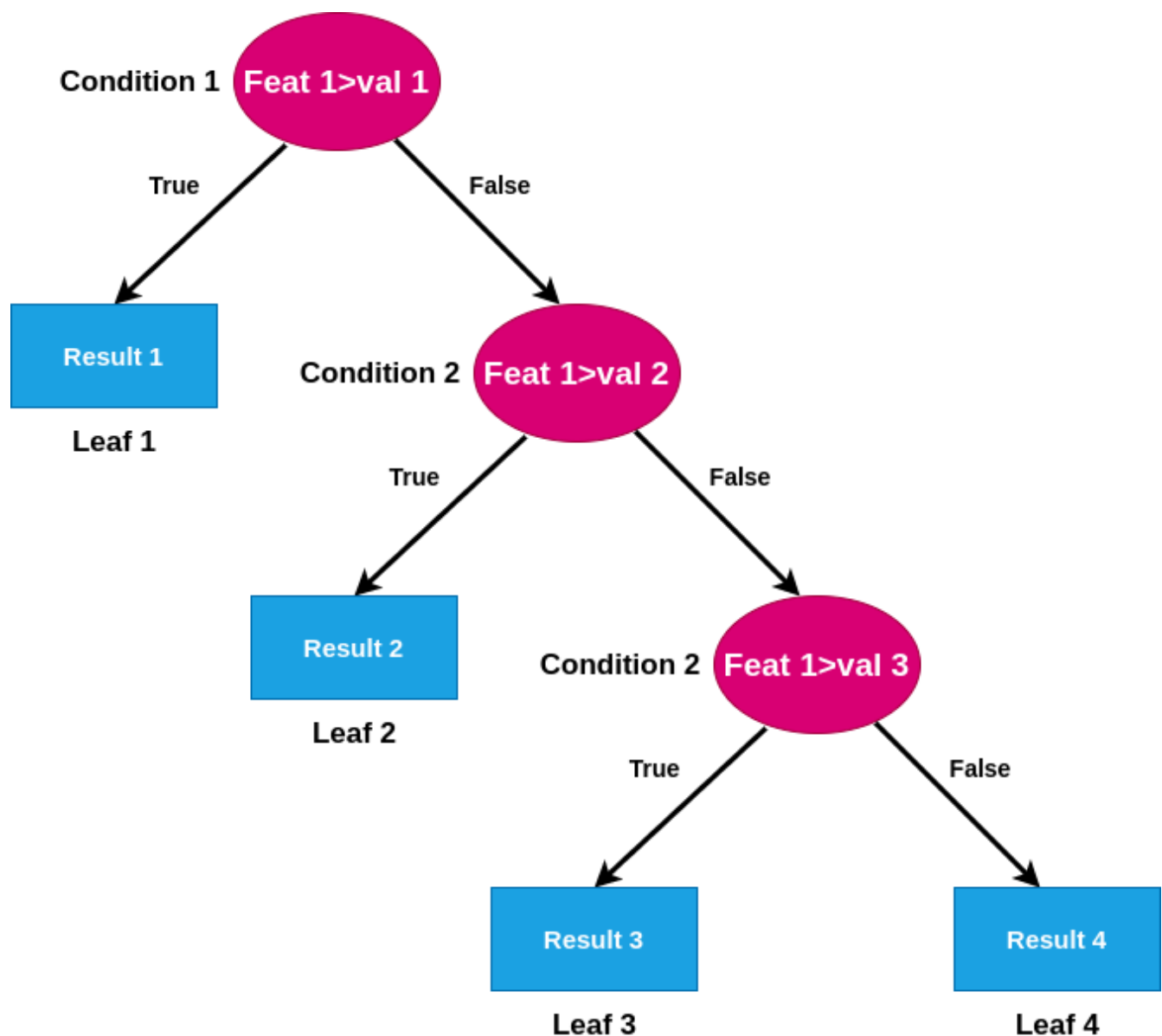
Let's consider the two conditions:



In the first distribution, we can see, we can easily fit a line to it. So, in this case, we can use linear regression to predict a value. But, in the case of the second distribution,

it is very evident that we can't fit any particular straight line to the distribution to predict a value, as the distribution behaves very differently in different ranges. In other words, in the second case, the distribution is non-linear. In such non-linear cases, Regression Trees are used. A point of argument can be that we can use polynomial regression for non-linear data, but sometimes distributions are pretty complex to analyze and decide the polynomials so, the decision tree's condition-based approach is preferred.

In regression trees, the leaves represent a continuous numeric value in contrast to classification trees which normally represent boolean or discrete values at the leaves.

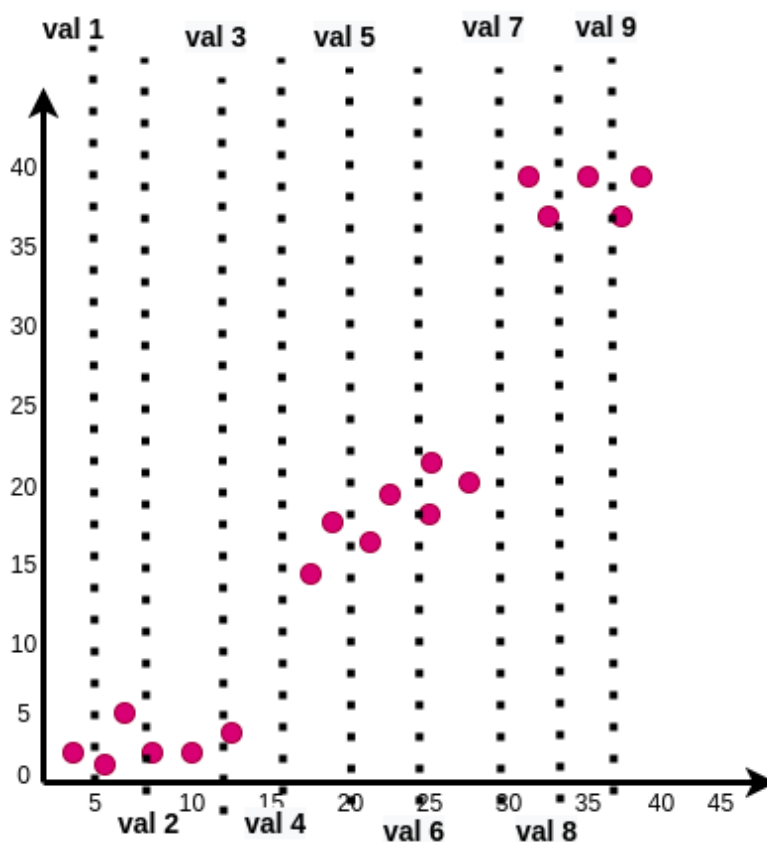


The above diagram represents the basic structure of the regression trees. The tree grows more complex and difficult to analyze when multiple features chip in and the

dimensionality of the feature set increases.

Now, let's see how we decide which value we should pick for creating the conditions. It is a bit different from what we did in the case of classification trees.

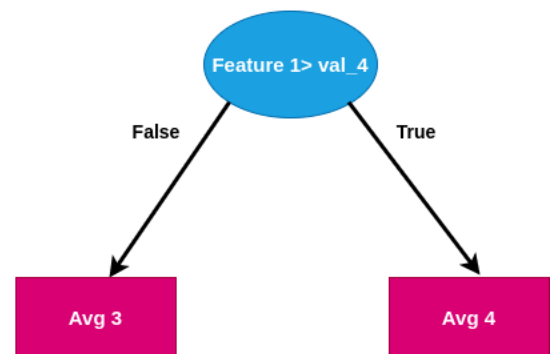
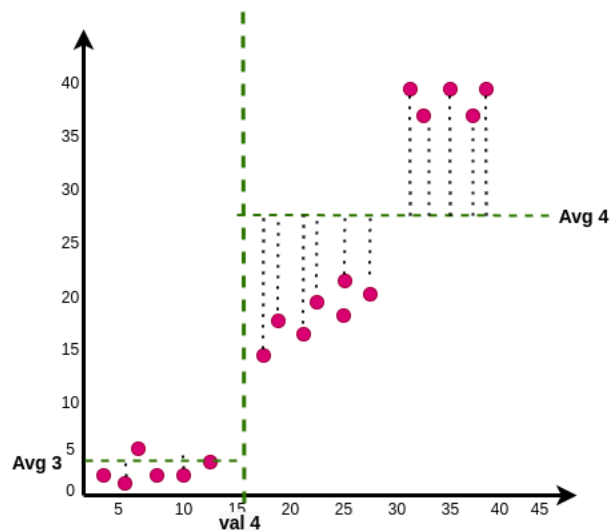
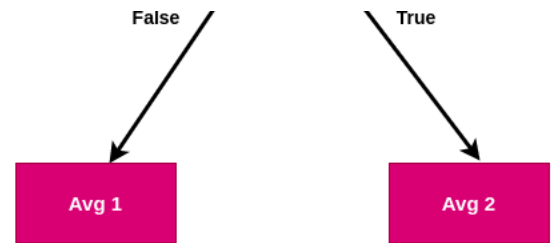
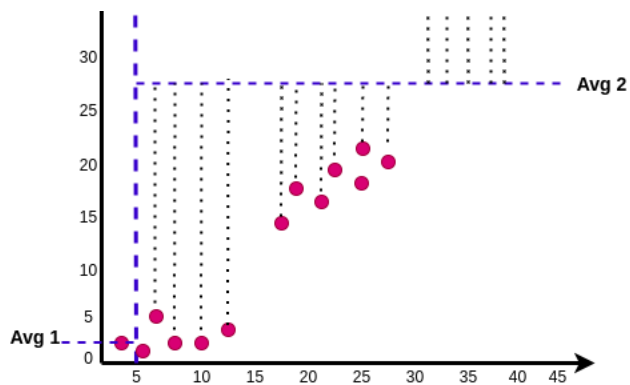
Let's say we have a feature set comprising of three features: Feature 1, feature 2, and feature 3.



Distribution for Feature 1

Let's consider the above image as our distribution for feature 1. Y-axis has the values on which we need to fit the regression and the X-axis has the feature values. So, we consider the values shown in the diagram as Val 1, Val 2, and so on, as the cutoff values to form the condition on Feature 1. These values are nothing but the average of feature 1 values of two corresponding points. Now, let's see how the calculations move forward.



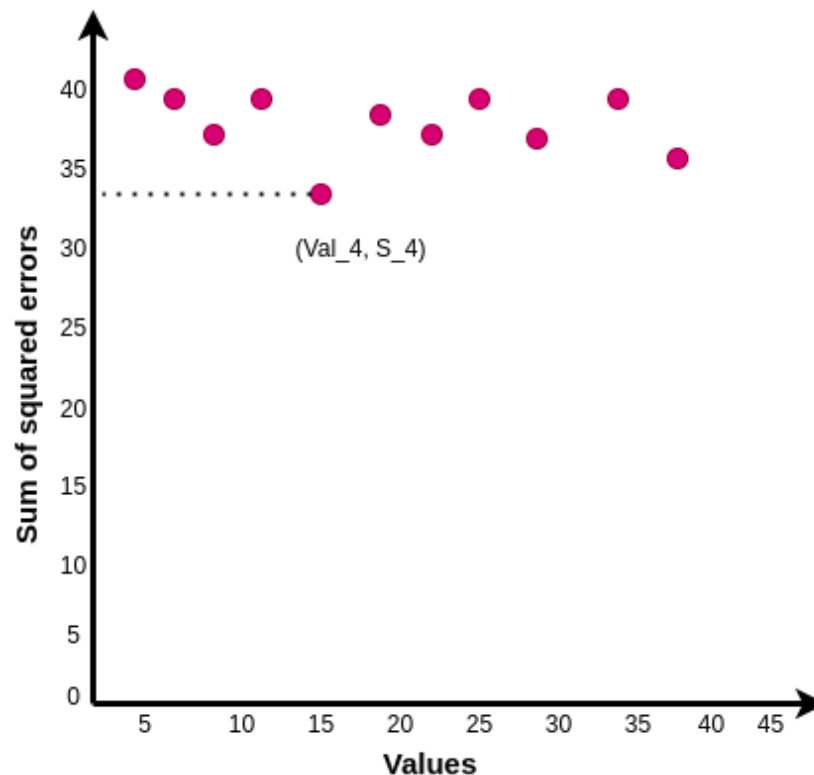


Now, let's consider the above diagrams. The diagrams are not to scale, so avg_2 is not equal to avg_4. When we consider the val_1 as the cutoff value for feature 1, we take the average values of all the Y-values for the points to the left of the line as avg_1, and similarly, we take the average values of all the points to the right of the line as avg_2. So, for any value, for which the condition feature 1 > val_1 holds True, the tree provides, the avg_2 as the prediction and vice versa.

Next, we take the squared errors for all the points. So for any point to the right of the line, the squared error of the point is $(\text{actual value} - \text{predicted value(i.e, avg_2)})^2$ and similarly for any point on the left, the squared error of the point is $(\text{actual value} - \text{predicted value(i.e, avg_1)})^2$. So, we find the squared error for all the points and sum them up. This is called the Sum of Squared Error. Say, for value 1 the Sum of Squared error is S_1.

Similarly, for value 4, the sum of squared error is calculated to say S_4. We can observe that S_4 will be much less than S_1. So, we find the sum of squared errors for

all the values we looked at, to be the cutoff to create the condition. If we plot them, we will obtain the plot as shown below.



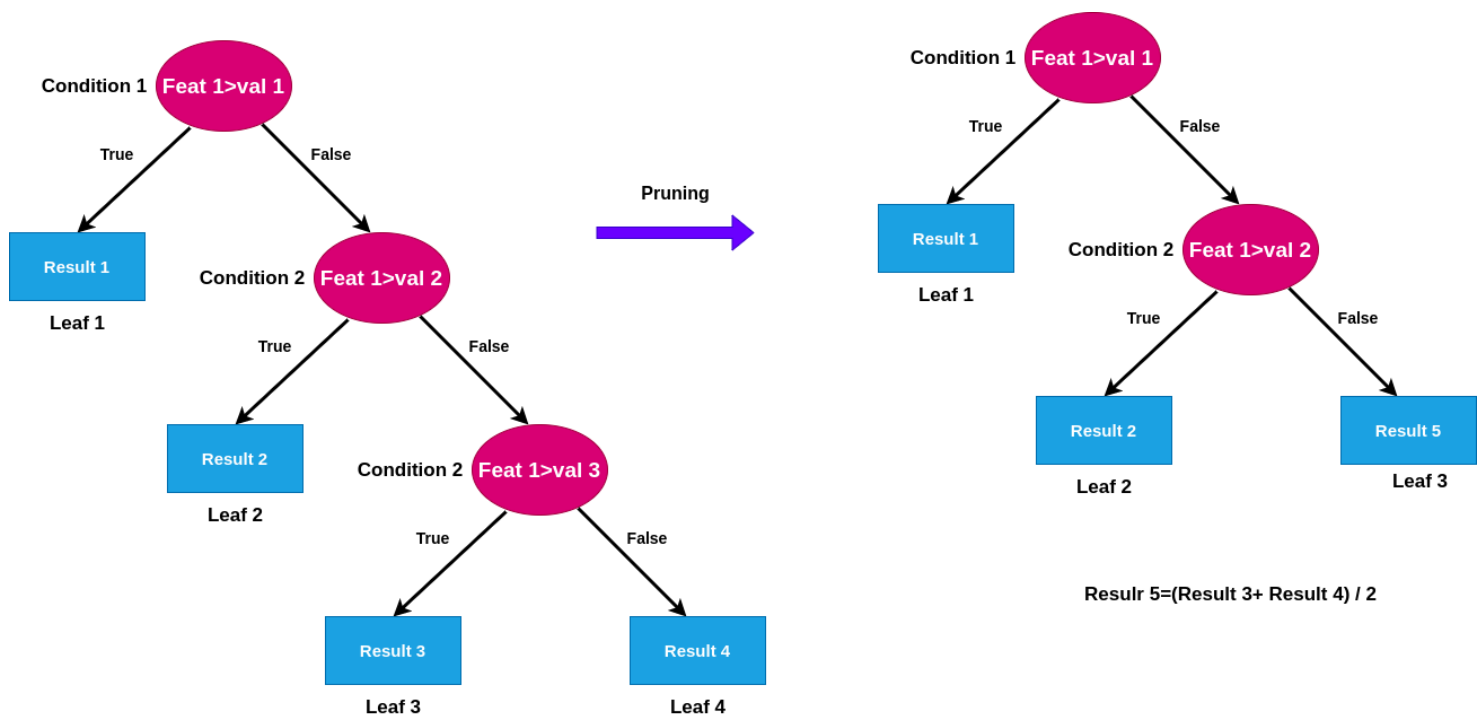
In the above diagram, we can see that the val_4 for Feature 1 provides the minimum sum of squared error for the point. So, now we have seen how the cutoff value for a particular value is decided.

Now, if we have multiple features, how to decide which feature should we use to create the node. For this, we create candidates. For instance, say for feature 1, val_4 gives the minimum sum of squared error. So, it is termed as the candidate for feature 1. Similarly, we find candidates for feature 2 and feature 3. Say the SSR corresponding to feature 2's candidate is S_2_3 , and that of feature 3's candidate is S_3_5 . We will compare S_4 , S_2_3 , and S_3_5 and select the minimum. The feature corresponding to the minimum will be used to create the deciding node.

The above-described method is used recursively to create Regression Trees. So, with this, we have seen how the CART mechanism works and how the classification and regression trees are formed. Now, briefly go through a very important concept in trees: Pruning.

Pruning

The method is used in trees to reduce overfitting. It is a procedure in which nodes and branches of trees are reduced in order to reduce the depth of the tree.



The above diagram shows how the concept of Pruning works.

There are basically two types of Pruning:

1. Pre-Pruning
2. Post Pruning

In Pre-pruning, we set parameters like 'min_samples', 'max_depth', and 'max_leaves' during the creation of the tree. All of the parameters need hyperparameter tuning and found using cross-validation and grid-search methods. It restricts the tree to a certain depth or a certain number of leaves.

Post-pruning methods are mostly done after the tree is already formed. Cost complexity pruning is the most used post-pruning approach.

Conclusion

Decision trees carry huge importance as they form the base of the Ensemble learning models in case of both bagging and boosting, which are the most used algorithms in

the machine learning domain. Again due to its simple structure and interpretability, decision trees are used in several human interpretable models like LIME.

In this article, we have seen how decision trees work using the principles of CART.