Decision Tree Classification in Python

1. Abstract

In this paper we will introduce, describe the working principle and code Decision Tree Algorithm. We will apply the algorithm on the problem of classifying the iris flowers based on some features like sepal and petal dimensions. We will devide our available data on training and test datasets, build our model on training dataset and test the accuracy our model with the test dataset.

2. Introduction

...

3. Decision Trees (DTs)

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression.

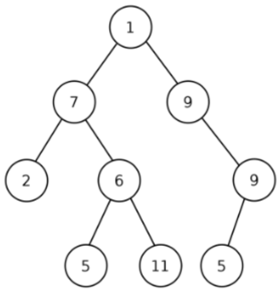
What is classification?

Classification is a process related to categorization, the process in which ideas and objects are recognized, differentiated and understood. Classification can also be defined as the grouping of related facts into classes. [1]

What is regression?

Regression is defined as a statistical method that helps us to analyze and understand the relationship between two or more variables of interest. The process that is adapted to perform regression analysis helps to understand which factors are important, which factors can be ignored, and how they are influencing each other. In regression, we normally have one dependent variable and one or more independent variables. Here we try to "regress" the value of the dependent variable "Y" with the help of the independent variables. In other words, we are trying to understand, how the value of 'Y' changes w.r.t change in 'X'.

As the name tells the structure of this model is tree data structure. In most cases it is a binary tree structure in which each node has at most two children, which are reffered to as the *left child* and the *right child*.



[1] https://en.wikipedia.org/wiki/Classification

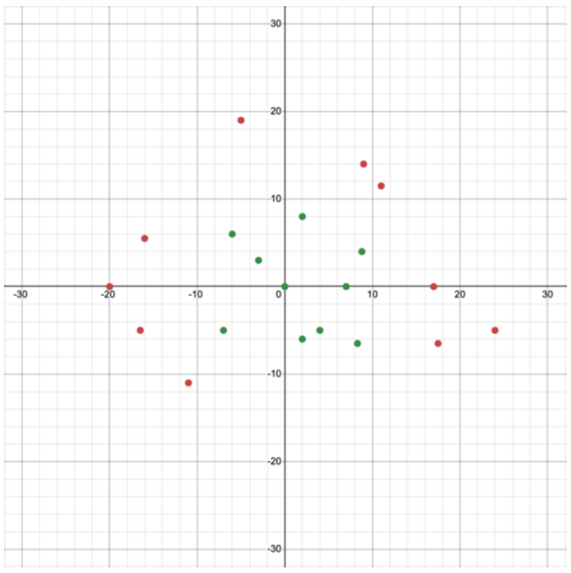
In computing, binary trees are used in two different ways:
1. First, as a means of accesing nodes based on some value or label associated with each node. Binary trees labelled this way are used to implement binary search trees and binary heaps, and are used for efficient searching and sorting.
2. Second, as a representation of data with a relevant bifurcating structure. In such cases, the particular arrangement of nodes under and/or to the left or right of nodes is part of the information(that is, changing it would change the meaning).

Now, the goal here is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation, but essentially it is a machine learning model or algorithm that uses a set of rules to make a decisions, similarly to how humans make decisions.

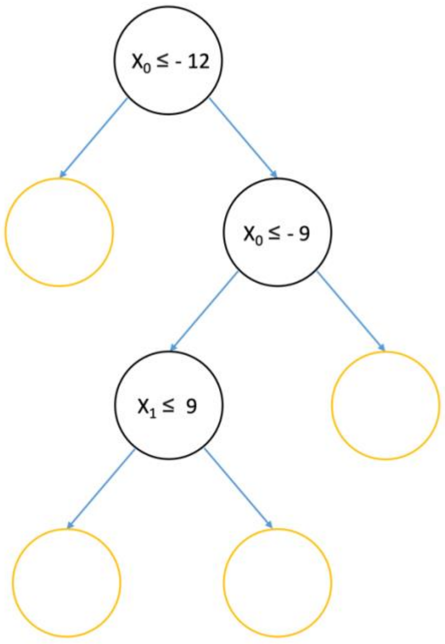
Tree models where the target variable can take a discrete set of values are called classification trees - in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.

4. The working principle of DTC

In order to expalin the working principle behind DTC we need some data, because no algorithm can learn without any previous experience. This dataset contains two features, X_0 and X_1. I have plotted X_0 along the horizontal axis and X_1 along the vertical axis.

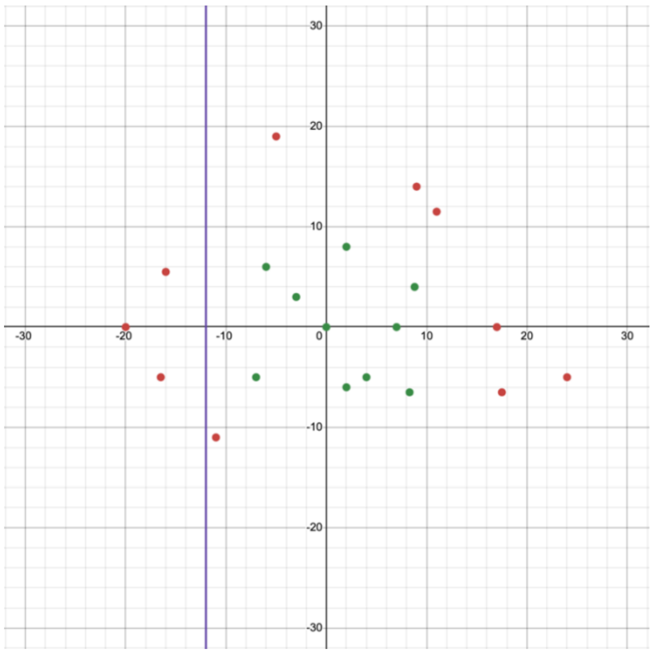


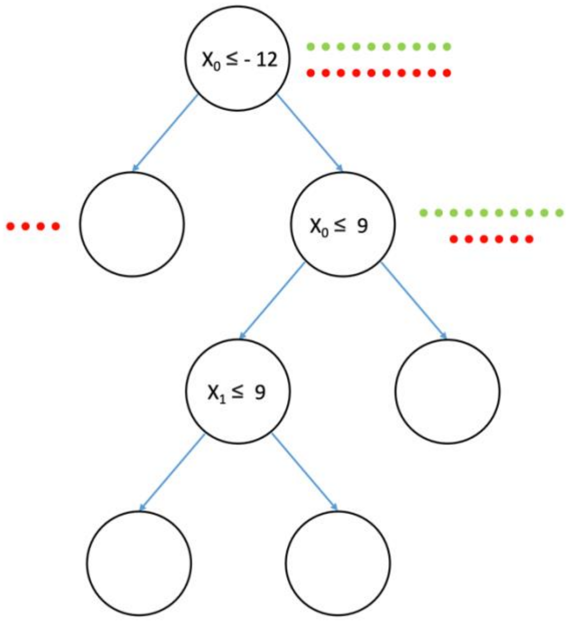
There exist two classes, green and red. If you notice carefully the classes are not linearly seperable that is you can't just draw a line to seperate them. This is intentional because here we want to represent a true power of decision trees. So what is a decision tree? Generally speaking it's a binary tree that recursively splits the data sets until we are left with pure leaf nodes, that is the data with only one type of class. Best way to start is to look at the trained decision tree. Below we can see the decision tree classifier for this specific dataset.



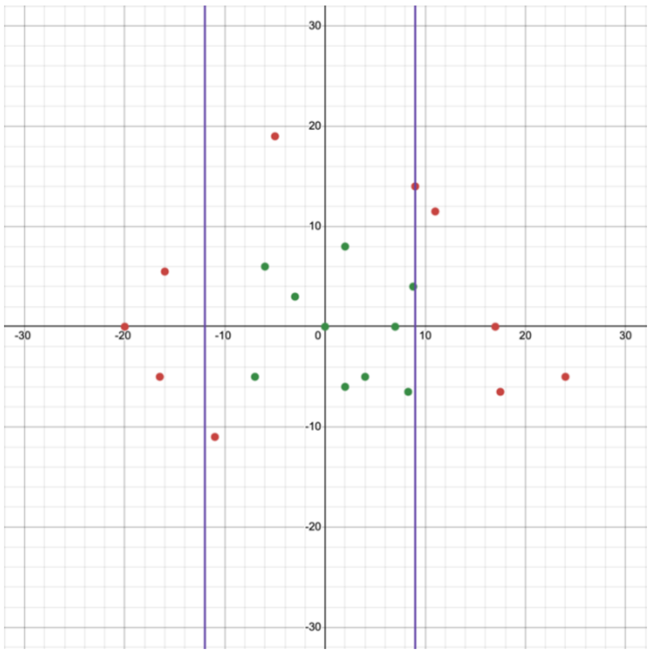
Notice that there are two kinds of nodes. Decision nodes (black) and leaf nodes (orange). The first one contains a condition to split the data and the second one helps us to decide the class of a new data point. How? We will see that later. For now we want to walk through tree first.

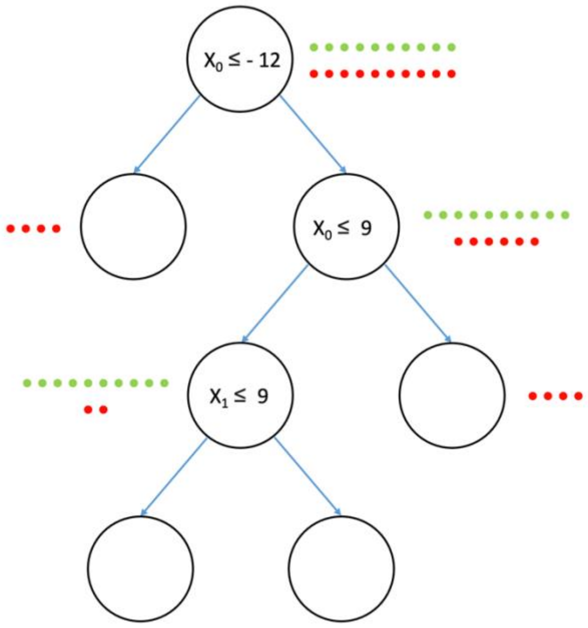
First, let's have the whole data at our root node. Based on the condition of the root node, we are going to split it and the condition is wether the X_0 feature is less than or equal to -12. In the plot the splitting condition looks like this purple vertical line. Every point that lies either left or on the line satisfies this condition. We are going to place those points in the left children of the root node and the points that don't meet the condition will go to the right children. We are going to follow this rule for the remaining nodes also. Please notice that the left child contains only red points, which means it's a pure node that we don't need to split any further.



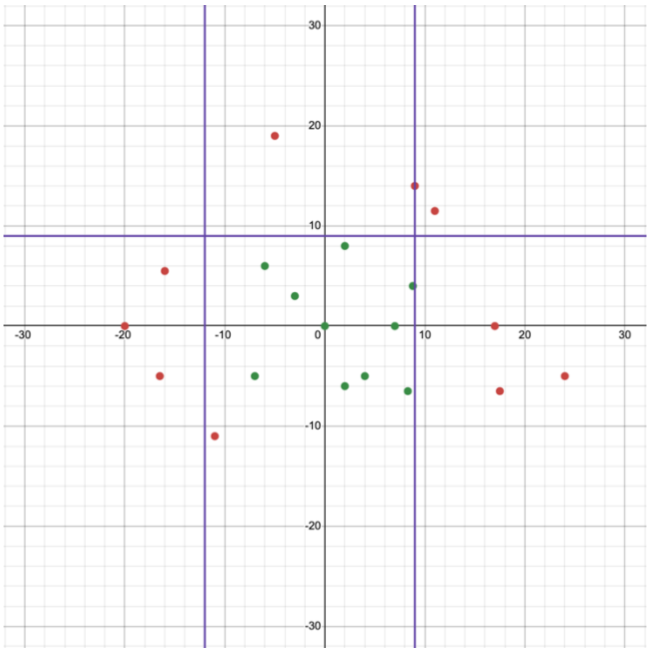


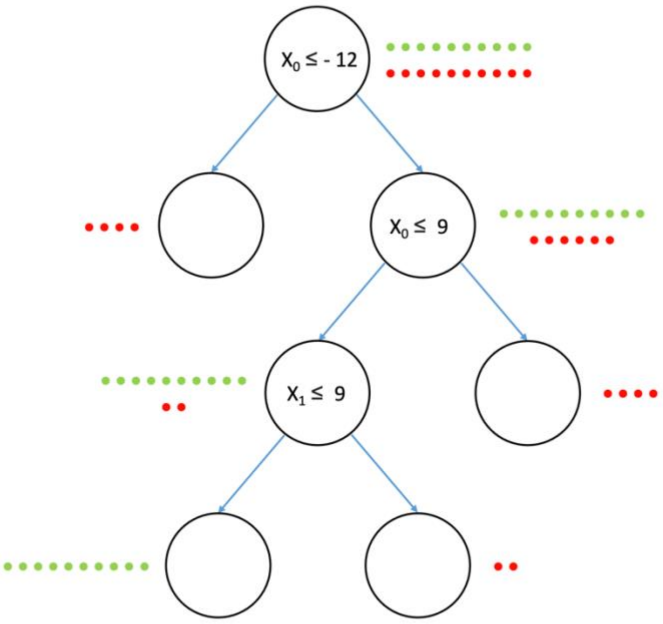
Let's now come to the right child. The condition is: $X_0 \leq 9$. Below you can see the splitting line and resulting child nodes. In this case the right child is pure so we are going to further split the left one.





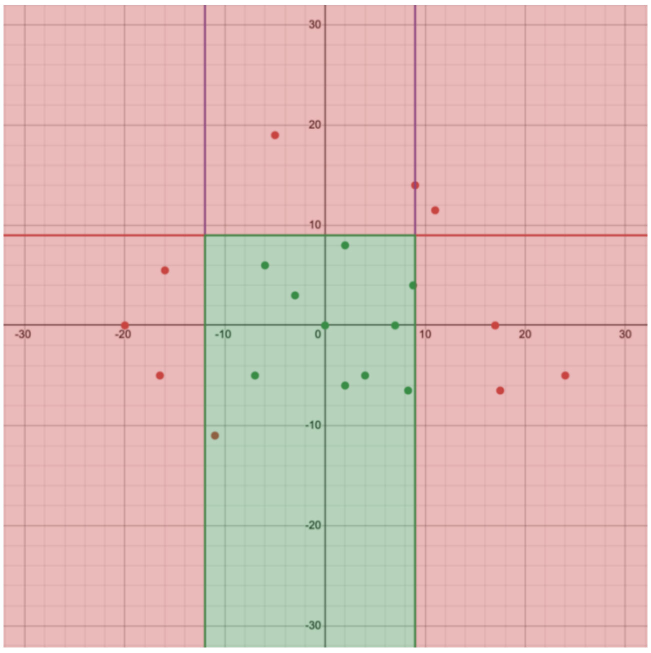
The third condition is $X_1 \leq 9$. Here we have a horizontal splitting line. The splitting looks as in an image below. Notice that in this case both the child nodes are pure thus we don't have to split any further.



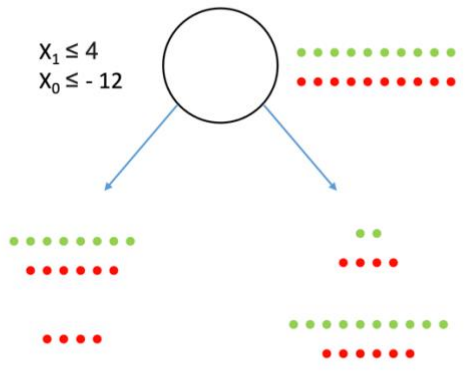


Now, the natural question is how can we use this tree to classify a new data point. Suppose that a new data point has X_0 and X_1 as 15 and 7 respectively. First we check condition at the root nood and since 15 is greater than -12 our new data point doesn't satisfy this condition therefor we come to the right child. Now the condition is if the X_0 is greater than 9, and since this is false too we again go to the right child which is a terminal leaf node that is node with all pure data. Since we are in the leaf node with red points only, we can classify our new data point as a red point. So that's how this algorithm helped us to assign a class to new data point.

The example we have explained here was pretty simple, where our terminal leaf nodes were pure nodes with the points of one class only. This is normally not the case for more complex data, where we rarely get 100% pure leaf nodes. In that case we perform a majority voting and assing a majority class to the test point. Let's also see how our model has devided the feature space into two regions based on the splitting criteria.



So, this is basically how we use a binary tree for the classification. At this point some might be thinking: this is just a bunch of nested if statements, so why do we consider this a machine learning. The answer is: Yes this model consists of nested if statements, but we need to know the correct conditions for splitting the data since there are many possible splitting conditions. Our model needs to learn which features to take and correspoding correct threshold values to optimally split the data. This is way it is called machine learning.
But now another question arises: How does our model decides the optimal splits. For explaning this let's start at the root. At this stage we have the whole data with us. Here we are going to compare two splits: $X_1 \leq 4$ and $X_0 \leq -12$. In the image below we can see the results of our splits.

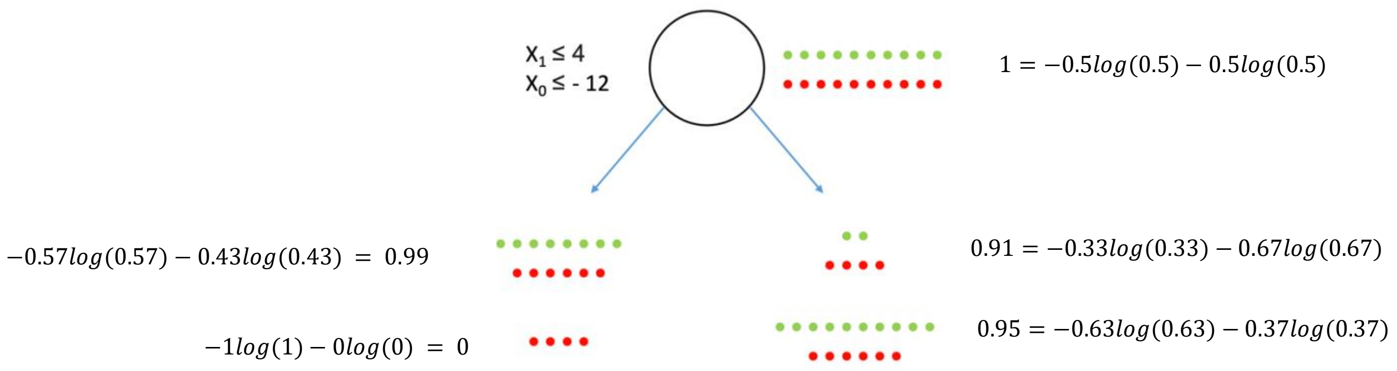


Now, which split would you choose? If we remember that our goal that is to get pure leaf nodes we must go for the second split because with this split we have successfully produced the left child with red points only. But how can our computer do the same, how can computer make this decision? The answer lies in information theory, more precisely the model will choose the split that maximizes the information gain. To calculate the information gain we first need to understand the information contained in a state. Let's look at the root state, here the number of red and green points is the same. Image if we are predicting the class of a randomly picked point, only half of the time we will be correct which means that this state has the highest impurity or uncertainty. The way to quantify this is to use entropy. Entropy is the measure of information contained in a state.

$$Entropy = \sum_i - p_i \cdot log_2(p_i) \tag{1}$$

$$p_i = probability\ of\ class\ i$$

If the entropy is high then we are very uncertain about the randomly picked point and we need more bits to describe the state. p_i denotes the probabilty of the ith class, and in the root state both the red and green points have the probablity of 0.5 and if we plug in the values we get that entropy corresponding to the root state is 1, which is the highest possible value of entropy. Using the formula (1) we will calculate entropy of the remaining 4 states. The result of this is shown in the picture below.


$X_1 \leq 4$
$X_0 \leq -12$
$1 = -0.5log(0.5) - 0.5log(0.5)$
$-0.57log(0.57) - 0.43log(0.43) = 0.99$
$0.91 = -0.33log(0.33) - 0.67log(0.67)$
$-1log(1) - 0log(0) = 0$
$0.95 = -0.63log(0.63) - 0.37log(0.37)$

We can notice that the state with the 4 red points gives the minimum entropy, that's why we call it a pure node. Now to find the information gain corresponding to split we need to subtract the combined entropy of the child nodes from the entropy of the parent node. The formula below describes this action.

$$IG = E(parent) - \sum_i w_i \cdot E(child_i) \tag{2}$$

$$w_i = weight\ of\ child\ i\ (relative\ size\ of\ a\ child\ with\ respect\ to\ the\ parent$$

Let's now calculate the information gain:

$$IG_1 = 1 - \frac{14}{20} \cdot 0.99 - \frac{6}{20} \cdot 0.91 = 0.034 \tag{3}$$

$$IG_2 = 1 - \frac{4}{20} \cdot 0 - \frac{16}{20} \cdot 0.95 = 0.24 \tag{4}$$

Clearly the second split gives greater information gain and that's why we would choose the condition $X_0 \leq -12$. Remember that here we have only compared two splits to show the working whereas in reality the model compares every possible split and takes the one that maximizes information gain. That is the model traverses through every possible feature and feature value to search for the best feature and corresponding threshold. This will be more clear when we meet with the code.

One important point:
Decision Tree is a greedy algorithm because it selects the current best split that maximizes the information gain, it does not backtrack and change a previous split. So all the following splits will depend on the current one, which does not guarantee that we will get the most optimal set of splits. On the other hand greedy search makes our training a lot faster and it works really good despite of its simpilicty.

Coming back to our tree, the model will now find the best split on the right child and so on. After this process is done we will get our final model.