All About ML

All About ML — Part 5: Decision Trees

Dharani J Follow
Apr 11, 2020 · 8 min read

Have you ever had difficulty in deciding what to do in a situation? Well as human beings we have this amazing(mostly defective 😛) habit of taking a decision in split second with out thinking. But assume you start thinking about the outcome, then your thought process would be to analyse the situation and draw insights to take a decision. Now think about a machine, it is also capable of taking a decision in split second but also by thinking and analyzing. Let us understand how Machine Learning finds its application in decision making.

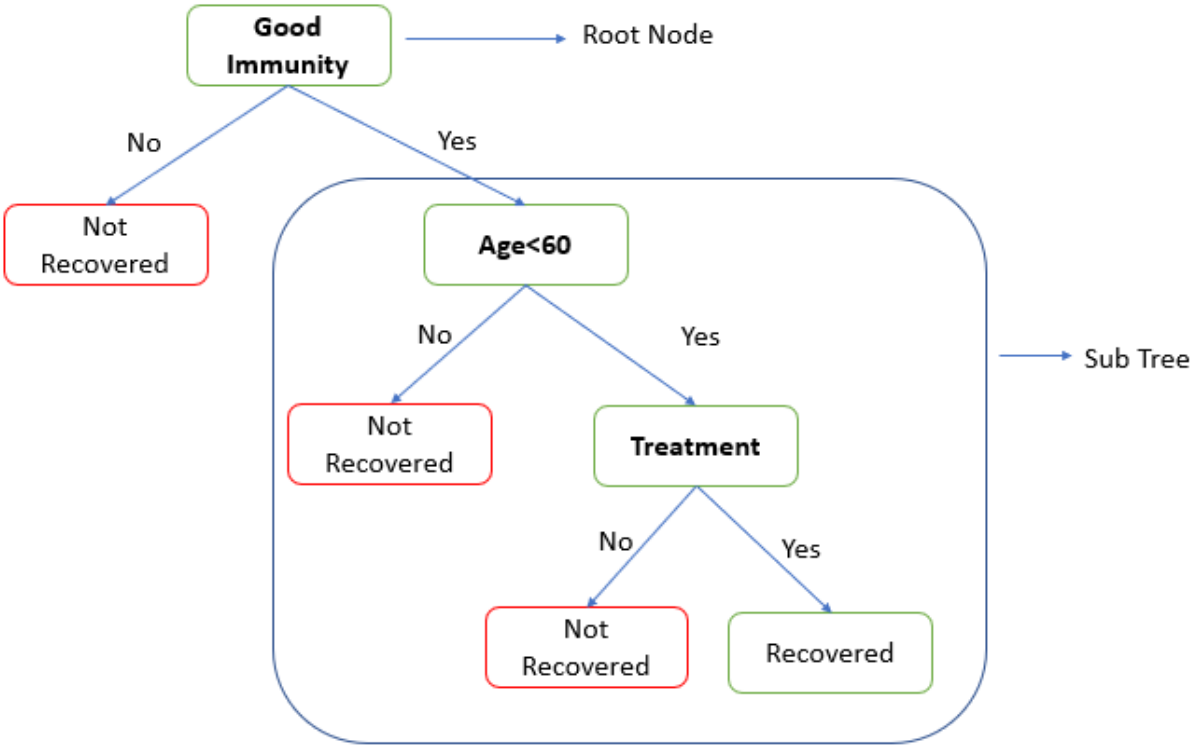

Fig 1: A simple Decision Tree to understand if a patient is recovered from a disease or not. (**classification tree**)

Predictor space(the whole data points of the independent variables) segmented into a number of simple regions. To predict a new data point, we assign it to a region in the predictor space based on the criteria it satisfies during the splits such that it will have the value as the mean of training observations(regression) or most commonly occurring class(classification) in that space. These kind of splitting rules can be used to segment are summarized in a tree, then these type of approaches are known as Decision Tree methods.

Common terminology in Decision Trees:

Root Node: The first condition where the whole data can be divided into two parts

Parent and Child Node: All the nodes in bold in Fig.1 are parent nodes i.e., which can further be divided. The further divided nodes(Sub-Nodes) are child nodes.

Leaf Node: In Fig. 1 the nodes which are Recovered/Not Recovered are leaf nodes or terminal nodes as the decision conditions are terminated with them.

Decision Node: All parent nodes except root node are decision nodes.

We can find the Decision Tree applications in Regression and Classification problems.

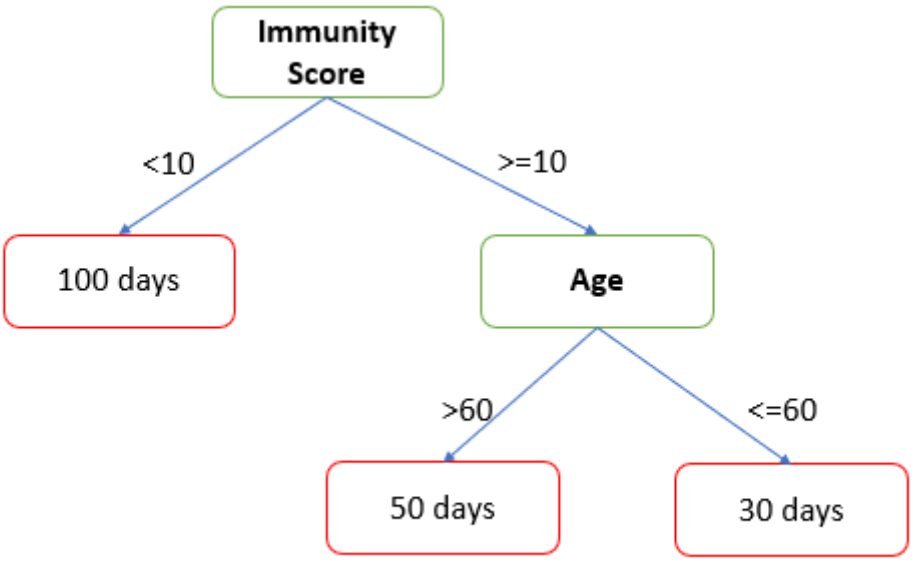

Fig 2: A regression tree with two features deciding on the approx numbers of days a patient takes for recovery

Regression Trees:

If we have a linear data then we can use classic linear models, but if the data is non-linear and we need to perform regression on output variable then Trees do a better job. Let us understand on the data related to patient's immunity scores, age and the days taken for recovery. In the above figure, it is clear how we can divide or split the feature space into different subsets(regions). If we can imagine it on a plane of data points, it should look like:

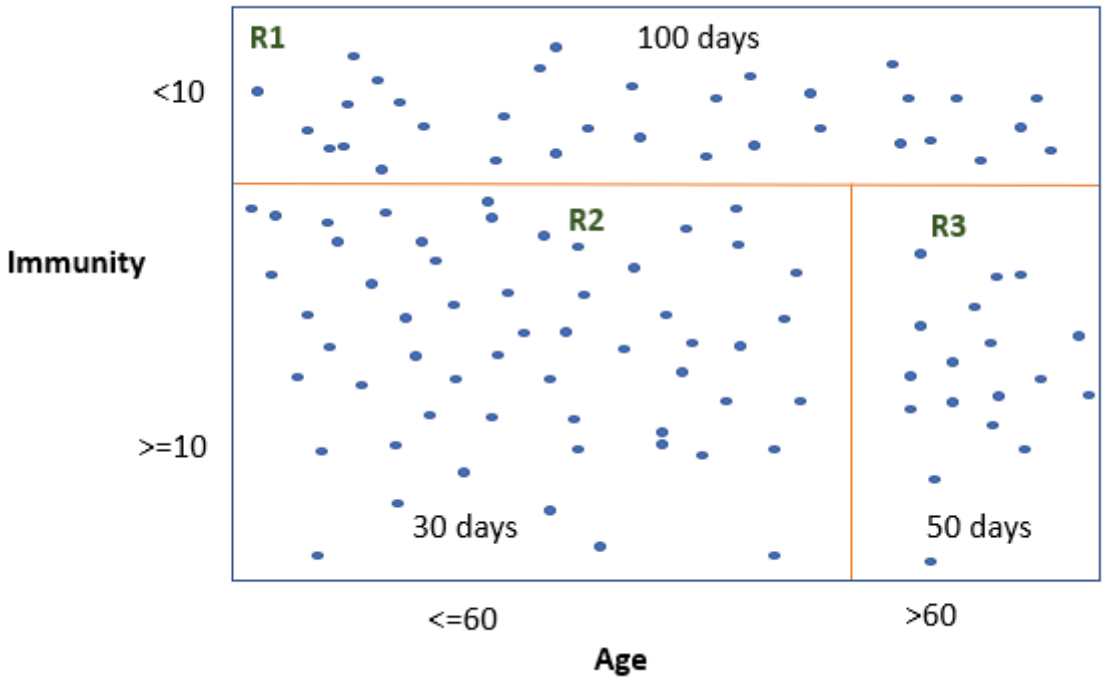

Fig 3 R1, R2, R3 are the regions this feature space is divided based on the split conditions.

As we see all the data points are divided into 3 regions based on the split conditions. So if we know the immunity score and age of a new patient, the algorithm places the new data point in one of these 3 regions and decide the approximate recovery days.

The example shown is having only two features so it can be seen on 2D plane for Fig-2. As the features increase, the visualization gets challenging.

Let us understand a little math behind data splitting into regions:

— If we are talking about the regression problem, we divide the whole data points into J **distinct** and **non overlapping** regions R1,R2,R3,…,RJ.(in our example — 3 regions)

— For every observation that falls into the region Rj, it assigns the same value for unseen data point. for example in Fig 2 we take mean of recovery days in each region and assign this mean to the new observation that falls in.

The challenge lies in the conditions to be picked for splitting into regions R1, R2,..,RJ. We have to divide them in such a way that we have to minimize the RSS,

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

Eq 1: ˆ yRj is the mean response for the training observations within the jth region

For this we follow the **top-down** approach. Top-Down approach is nothing but the split starts from a feature and goes on with next ones(an optimal feature which can divide the whole data into 2 or more homogeneous sets). This is also called **greedy** approach as the split starts from the beginning rather than waiting and picking a split that will lead to a better tree in some future step.

The algorithm starts finding the best split by **minimizing RSS** with in each of the new regions. But in next step it proceeds with one of the identified regions and apply then same steps on it, now we have 3 regions. Then the split happens on these 3 regions by minimizing the RSS and so on. This process reaches an end with a stopping criteria like depth is 5 or 6 based on the problem statement.

Tree Pruning:

If we have many regions then the algorithm have understood and split the train data perfectly. But when a new data point comes in there is a high probability of **overfitting**. So it is advisable to have fewer regions with low variance and low bias such that it can predict test data with the same accuracy as train data which should be the ideal case. But how do we know how many regions or splits is needed to achieve the ideal conditions? Here comes the concept of **Tree Pruning**.

In pruning, the strategy is to first split into many regions so that we obtain a very big tree lets say To (T not) and then prune it back to obtain smaller subtree with minimal error rate. This process of pruning by reducing error is called **Cost Complexity Pruning**. As we are not sure which subtree will be a correct one to prune back, we go by math intuitively. We introduce a non negative tuning

parametre **α(alpha)** such that a sequence of trees form a subtree T that is subset of To such that the equation shown below is as small as possible.

$$\sum_{m=1}^{|T|} \sum_{i:\ x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

Eq 2: Cost Complexity Pruning

Here |T| indicates the number of terminal nodes of the tree T, Rm is the region (i.e. the subset of feature space) corresponding to the mth terminal node, and ˆyRm is the predicted response associated with Rm — that is, the mean of the training observations in Rm. The tuning parameter α controls a trade-off between the sub tree's complexity and its fit to the training data. When $\alpha = 0$, then the sub tree T will simply equal To, as it is simply Eq 1 shown and is just training error. However, as α increases, there is a price to pay for having a tree with many terminal nodes, Eq 2 tends to minimize for a smaller subtree. We pick alpha using **Cross-Validation** technique.

> *If you are familiar with Lasso regularization, Eq 2 is similar to it as we curb the complexity of linear model using Lasso. (My blog on Lasso for reference if needed)*

Classification Trees:

These are similar to Regression trees but instead of taking the mean of observations in the leaf node or region, we calculate the highest number of occurrences of a class of that target variable in that region. In regression we take RSS as the criteria for splitting optimally, however in this scenario the classification error rate is simply the fraction of the training observations in that region that do not belong to the most common class.

$$E = 1 - \max_k(\hat{p}_{mk})$$

Classification error rate

ˆPmk represents the proportion of training observations in the mth region that are from the kth class. There are two more metrics which are used widely other than E, to calculate the classification error. They are **Gini Index**:

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

Gini Index

G defines the total variance across K classes. It measures the degree or probability of a particular variable being wrongly classified when it is randomly chosen from the data set. It is also said Gini Index is the **measure of Purity** of that node so if a small Gini Index says that the node is Pure with majority of the predictions belongs to a single class. Another measure is **Cross-Entropy** which is:

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

Cross Entropy

Since $0 \leq \hat{}$ Pmk ≤ 1, it follows that $0 \leq -\hat{}$Pmk log ˆPmk. If ˆPmk is near to zero or near one, Cross-Entropy will take a value near zero and it implies the node is pure. Cross Entropy is also similar to Gini Index and it used to understand the **disorder of a grouping** by the target variable.

So E, G and D are used as metrics to understand the quality of a split at each node and help in Pruning.

Advantages of Decision Trees:

1. Easy to understand and explain
2. Works really good on smaller data sets and can be viewed graphically
3. With out creating dummy variables, they can perform well on classification problems
4. Computationally fast in classifying unknown data points
5. Inexpensive in terms of space utilization

Disadvantages:

1. Trees can be non robust and overfit. A small change in data can cause large impact on predictions.
2. Large tress often unable to predict and the results may not match the expected.
3. If a node is having many splits then there is a possibility of giving more importance to that hence resulting in biased predictions.

In python we have Decision Trees model in **sklearn** package each for Regression and Classification. Let us understand the important parametres that can be modified to achieve better results.

criterion: Defines the error functions we want to use to obtain the quality of split at each node.

Regression- {default= “mse”, “mae”, “friedman_mse”}

Classification- { default=”gini”,“entropy”}

max_depth: The maximum depth of the tree.

min_samples_split: Defines the minimum number of samples that are needed to split and internal node in the tree. default = 2

There are few other parametres in the model but these 3 are mainly used in shaping the model for the need. The other parametres include **min_samples_leaf, min_weight_fraction_leaf, max_leaf_nodes** etc.,

We find numerous applications in daily life of decision trees it is important to understand the problem statement precisely to know use the parametres in the model to get desired results.

References: An Introduction to Statistical Learning: With Applications in R

Thank you!

Machine Learning Decision Tree Gini Index Cross Entropy Pruning