Decision Trees

Mads Møller

LinkedIn: https://www.linkedin.com/in/madsmoeller1/

Mail: mads.moeller@outlook.com

This paper is the fourth in the series about machine learning algorithms. Decision Tree algorithm can both be used for *classification* problems as well as for *regression* problems. In this paper we will focus on how decision threes can be used for classification problems. The Decision Tree is also a supervised machine learning algorithm, meaning that our data need to be labelled. The decision tree algorithm is one of the most popular algorithms in machine learning. One of the reasons that decision trees has become so popular is since they are rule-based and visual. It is moreover a multi-class classification algorithm.

1 Intuition behind Decision Trees

Overall, Decision Trees are build as a lot of if-else statements. If you are familiar with if-else statements in general, decision trees will be easy to understand. Let us take a simple example to understand how decision trees classify data:

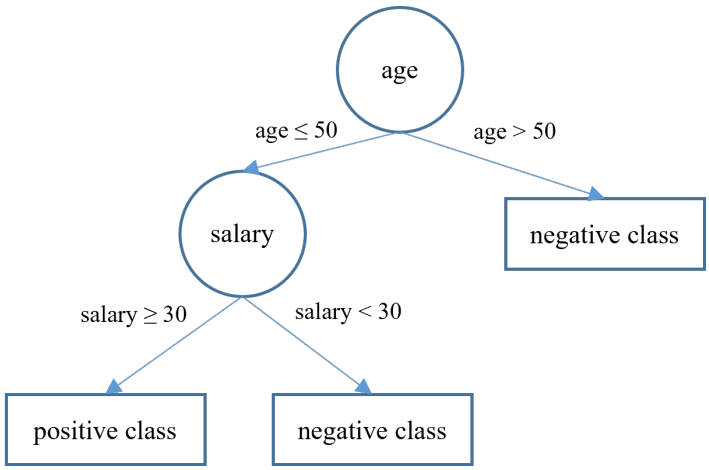

Figure 1: Decision Tree Strucuture

If we take a look at figure 1 we have an underlying dataset of two explanatory variables, *age* and *salary*, and a class variable which is *loan* where we have two classes **approved** and **declined**. The way Decision Trees work is that it uses the concept of *divide and concur*. The node in the top of figure 1 is called the **root node**, because it contains the whole dataset. The three nodes which contains the classifications

are called **leaf nodes**. From all nodes, except for leaves, there are branches coming out. A **branch** is defined by 'if' statements. Likewise all nodes, except leafs, has **children nodes** which are nodes coming out of another node, called the **parent node**. What we now need to understand is how decision trees are making these if-statements.

2 How are Decision Trees built?

First, we need to understand the *splitting rule*. The splitting rule is about splitting a node (parent node) into two other nodes (children nodes). A splitting rule requires an explanatory variable and a threshold of classification. Just like as in figure 1. To understand how nodes are split we need to understand the term **impurity**.

p_i, describes the proportion of observations in class i. Impurity is a measure of how much information is needed to classify. We say that a node is pure if it only contains observations from one class. But how does we measure impurity? We can use *entropy* (like in logistic regression) and the *gini index*. Mathematically we can write Entropy as:

$$\text{Entropy} = -\sum_{i=1}^{n} p_i \cdot \log(p_i) \tag{1}$$

So how we can interpret impurity from an entropy definition is simple. Entropy must be the amount of information needed to accurately describe the data. If the data is homogeneous, meaning all the elements are similar, the entropy will be 0. If the data is equally divided the entropy will be 1. Let us also take a look at the gini index:

$$\text{gini index} = 1 - \sum_{i=1}^{n} p_i^2 \tag{2}$$

The interpretation are very similar with entropy. The gini index is a measure of inequality in data. But how exactly is a node split then?

The objective of the algorithm is to *maximize gain in purity*. To understand what this means let us consider a parent node m. m_1 and m_2 is the number of observations in child 1 and child 2. Therefore, $m = m_1 + m_2$. The objective (gain in purity) is then defined as:

$$\text{gain} = 1 - \frac{m_1}{m}\text{impurity}_1 - \frac{m_2}{m}\text{impurity}_2 \tag{3}$$

The way the algorithm finds the maximized gain in purity is with full enumeration. So it goes through all explanatory variables and tries each possible split for each variable. The split and variable which maximized the gain in purity will be chosen. Mostly the gini index is used for calculating the impurity. However, this splitting can continue until we only have pure nodes. But if the algorithm continued until this stage the model would be overfitted. We need to define a stopping criteria to tell the algorithm when to stop splitting parent nodes and make a classification.

The usual stopping criterion is simple. The number of observations in each node should be above a

minimum (this becomes a hyper-parameter in our tree. The last thing we need to understand is how a decision tree assigns a class to leaf node. We have to assign a class, y^l, to each leaf node l. Usually the class y^l assigned to a leaf node is the majority class in leaf node l.

2.1 Cost-Complexity Pruning

The goal is to minimize:

$$\text{error}(T, S) + \alpha \cdot \text{size}(T)$$

Where error(T, S) can be the misclassification error. size(T) is the number of leaf nodes. We can tune the hyperparameter α. If α is large the tree will be smaller. This is also a hyperparameter which can be tuned in the decision tree algorithm.

3 Random Forest

The Random Forest algorithm is another algorithm in machine learning. One is more likely to use Random Forest instead of the Decision Tree Algorithm. This is because it in general performs better than the Decision Tree algorithm. However, **Random Forest is actually just a collection of Decision Trees**. A single Decision Tree might be unstable and overfitted. However, imagine that we make 500 draws with replacement (bootstrapping) from our training data. we then have 500 different training folds. If we make a decision tree on each of the 500 training folds we now have 500 different decision trees. We have created a "forest" of decision trees. When the random forest algorithm classifies an observation it chooses the *mode* (the prediction that occurs most times) of the 500 trees. So, Random Forest uses a collection of results to make a final decision. Therefore, Random Forest belongs to a type of algorithms called **ensemble**. One of the best performing algorithms at the moment also belongs to the ensemble category, this is called *XGBoost* (or Extreme Gradient Boosting). We will have a separate paper on XGBoost.

3.1 Feature Importance

As a by-product of Random Forest one is actually capable of showing the importance of each explanatory variable in the model. Feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node. The node probability can be calculated by the number of samples that reach the node, divided by the total number of samples. The higher the value the more important the feature. You can read more about the calculations behind feature importance here. Feature importance is however an very interesting feature to show which lots of other machine learning algorithms are not able to.