CART IMPLEMENTATION FROM SCRATCH

Meritxell Arbiol and Gerard Sànchez

30-04-2022

ABSTRACT. In this paper we are going to implement the Decision Tree algorithm from scratch to learn and understand all the details about how it works. We will better understand some concepts such as the Gini Index or Entropy, how to create the splits in the tree and when to stop splitting. We are also going to evaluate how good is the classification obtained with this algorithm created by us making predictions using a dataset.
Finally we are going to compare the results obtained using our algorithm with those obtained using other algorithms already defined such as the function DecisionTreeClassifier of the sklearn library. We know that the results for each algorithm will be different due to small decisions made in each one but let's see how different the results are.

1 Introduction

In this project we are going to create from scratch a very used algorithm to make predictions called Decision Tree. The implementation will be created in python. It is a great opportunity to better understand all the little details that this algorithm has and how by making small modifications the result in the classification can vary significantly.

We will also be able to better understand how other algorithms already created such as the DecisionTreeClassifier of the sklearn library, which we believe will be a little different from those obtained with our implementation due to small decisions that have been made. This algorithm is known as Classification and Regression Tree Algorithm (CART), which we are going to focus on the classification part where we will use a dataset from the UCI repository, called "Banknote authentication Dataset".

2 What is a Decision Tree?

The Decision Tree algorithm, better known as Classification and Regression Tree, is a very famous algorithm to make classifications or regressions predictions creating models.

This algorithm is often used as a basis for other algorithms such as Random Forest or Boosted Decision Trees. This algorithm belongs to the supervised learning algorithms.

With this algorithm we can create models that will help us to predict a class or a value. This model has been created from training data which have made the model learn following a sequence of decision rules.

As the name of the algorithm itself indicates, we can imagine a tree, in which to predict a class, we start at the root, where we compare the value of an attribute of our data to be classified with the value of the root. Then, depending on whether the value is higher or lower than the value of the root, we will follow one branch of the tree or another to the next node. In this way we will repeat the same process recursively until we reach a leaf, which will be our predicted class.

Another thing to keep in mind for this algorithm is that depending on the type of target we have we can have two kinds of trees: Categorical or Continuous, and depending on this we are talking about classification or regression. When the target is categorical we are going to classify. When the target is continuous we are going to do a regression. As we have mentioned before, in the case of this project we are going to focus on a categorical target, so we are going to do a classification.

Thus, as we can see in the Figure 1, we work with binary trees. Each node represents a decision, where a variable (x) enters and there is a partition depending on the value of the variable. In the terminal nodes we have the leaves, which contain the output variable (y) which is the result of our prediction.

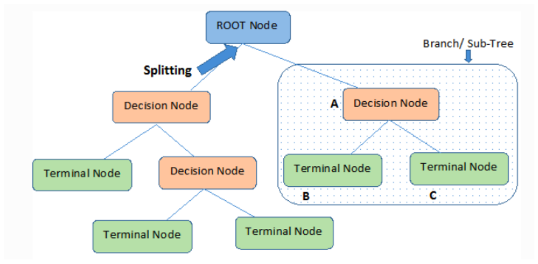

Figure 1: Binary tree

The tree can be represented as a graph or only as a set of rules. In a tree there are the terms parent and child. For example, in the case of the tree above in the Figure 1, we see that A is the parent of B and C (direct links in the tree), and nodes B and C are siblings.

We have special terminology when we talk about trees [1]:

- **Root Node:** is the first node in the decision tree where we have the entire population, and it will be divided into 2 groups after the decision is made.

- **Split:** Process of dividing a node into two sub-nodes starting from the root node.
- **Decision Node:** Node in which a decision is applied under which the dataset is going to be divided into two different branches.
- **Branch or Sub-Tree:** Subsection of the tree.
- **Pruning :** is a technique to reduce the size of the decision tree by removing sub-nodes of the decision tree. The aim is reducing complexity for improved predictive accuracy and to avoid overfitting.
- **Leaf or Terminal node:** Final node which contains the output variable of our prediction. Also known as Terminal Node.

3 Introduction to the main parts of a Decision Tree

To fully understand how a decision tree works, it is important to know the following parts in detail. Later we will use them to create our own implementation of the tree.

3.1 Impurity

The decision to make more or less splits affects the accuracy of the algorithm. For this purpose, several algorithms are used to decide whether to split a node in two or not. When data is divided into smaller chunks, the target column becomes more homogeneous. Increasing the homogeneity means that we could say that the purity of a node increases with respect to the node of the variable. So, we are interested in homogeneity, so what the Decision Tree algorithm does is to make all possible splits for all variables and then selects only those splits that have more homogeneous results in the sub-nodes.

We have to be careful not to obtain a perfectly homogeneous tree, as this implies that it is very complex (many branches and leaves). If a tree is perfectly homogeneous, it can happen that in the different leaves of our tree we have only a single record. This means that there is an overfit, this will have to be regularized (regularize means controlling the growth of a tree during the training) applying some restrictions that we will explain later.

Now comes the important question: how to decide the thresholds applied in each node to split the data in two different branches? Decision tree uses some mechanisms to reduce the impurity of the target column. We say that when a node has many classes it is impure, while when a node has one class it is pure. We have some measures to calculate the impurity of a node, some of them are Gini Index, Entropy, Chi-Square, Reduction in Variance, Gain Ratio among others, but for this project we are going to focus on Gini Index and Entropy which are the most used.

3.1.1 Entropy

Entropy is a way of measuring the impurity of a node. With entropy we can measure the degree of randomness of the data we want to process. Entropy can have values between 0 and 1, and the higher the value, the more difficult it will be to make predictions. This is logical since if something happens totally randomly there is no way to make predictions of what might happen.

The Entropy is defined as:

$$Entropy = -\sum P_i * Log_2(P_i) \tag{1}$$

As we can see in the Figure 2 [2], on the X-axis we have the probability and on the Y-axis we have the degree of entropy. We can see that when the probability is 0.5, the entropy level is maximum (1), and as we said before, this means that the randomness is maximum so it is more difficult to draw conclusions with this type of data. In the same way we see that when the probability is 0 or 1, the entropy will be 0 (uncertainty is 0) and predictions can be made without problem so it is favorable to us.

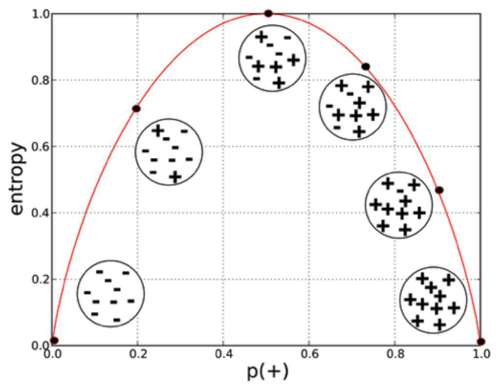

Figure 2: Relation between Entropy and Probability

Knowing what entropy is, we now need to know how it can be relevant during the decision making process within the Decision Tree algorithm. We know that the decision tree finds a threshold for a particular attribute given some data, so that when the algorithm applies a function on that attribute column, it will split the data into two nodes.

When sub-branches are created, what has to be controlled is that the entropy of the sub-branches is always less than the entropy of the parent branch. The more information gained the better. And to have a higher information gain, the difference between the entropy of the previous node and the current node has to be as large as possible.

3.1.2 Gini Index

Using entropy or the Gini Index will not significantly affect the results. Next we have the Gini Index formula:

$$Gini\ Index = 1 - \sum (P_i)^2 \tag{2}$$

Gini Index is a function that calculates the cost to evaluate the different splits in the dataset. As we can see in the formula below, Gini Index is calculated by subtracting the sum of the squared probabilities of each class from one. The n in the formula is the number of classes, and Pi is the probability choosing the class i for that data point.

Gini index works only to perform binary splits. The higher the value of the Gini Index, the higher the heterogeneity. And as we have mentioned before, we are interested in those splits that have a higher homogeneity in the results, so we want a small Gini Index value.

4 Building a Decision Tree

Once all the main parts of a decision tree, the impurity and its calculation have been explained, the process of splitting a dataset into a decision tree is detailed with the CART implementation which uses Gini Impurity. The goal is to determine which separation is the best for all of the features of the data, thus, we need a way to measure and compare the Gini Impurity in each attribute. It is done via the GINI gain which is the parent node Gini impurity subtracted by the weighted average of the Gini impurities of the left and right nodes. The highest Gini gain value on the first iteration will be the Root Node. The main process to create the best split is as follows:

1. Calculate the GINI gain impurity for each feature of the dataset.
2. Compare the different scores using a new attribute to separate data. If the new node has a lower score after splitting the data, there is no point in separating the data in the direction of that leaf.
3. If separating the data results in an improvement, then pick the separation using the variable with the highest gain and delete this variable from the list of features.
4. Generate a node with that variable (if it is the first iteration is the root node)
5. Repeat the process for each partition(leaves) of the values of the attribute.

After the process of splitting is exposed, let us define $\{\mathbf{x_i}, y_i\}_{i=1}^{N}$ the training dataset where x_i are the features and y_i the target variable with classes 0,1. Thus, it is defined the calculation of the GINI gain for a particular feature as follows:

- if $\mathbf{x_i}$ is categorical with classes 0,1 and p_{jk} the number of elements being class y_i=j and taking the value x_i=k, we calculate the average GINI of the node leaf $\mathbf{x_i}$ as:

$$I_x = \frac{n_0}{n} \cdot I_{left} + \frac{n_1}{n} \cdot I_{right} \tag{3}$$

where

$$I_{left} = 1 - (\frac{p_{00}}{p_{00} + p_{01}})^2 - (\frac{p_{01}}{p_{00} + p_{01}})^2 \tag{4}$$

and

$$I_{right} = 1 - (\frac{p_{10}}{p_{10} + p_{11}})^2 - (\frac{p_{11}}{p_{10} + p_{11}})^2 \tag{5}$$

Then, the Gini GAIN is defined as follows:

$$I_{GAIN} = I_{parent} - I_x \tag{6}$$

where

$$I_{parent} = 1 - (\frac{n_0}{n_0 + n_1})^2 - (\frac{n_1}{n_0 + n_1})^2 \tag{7}$$

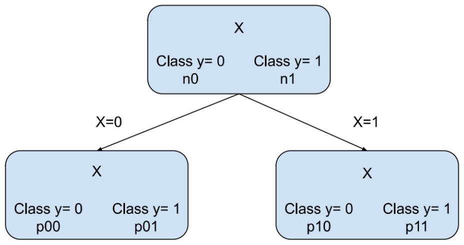

Figure 3: Representation of a Node for the calculation of the GINI index

- if $\mathbf{x_i}$ is continuous with values $\{v_1, ..., v_N\}$, it is needed to split and get the split value of the numeric variable. The algorithm search for the best split among numeric columns and the main steps are:

  1. Sort $\mathbf{x_i} = \{v_1, ..., v_N\}$ by ascending order.
  2. Calculate the average between neighours values of $\mathbf{x_i}$ and consider each value as the threshold for the split. For each different split, compute the GINI gain.
  3. The final split value m is the one with the highest GINI gain. Turn the split into the boolean test for the node Data will be split into $[x_i < m_i]$ and $[x_i >= m_i]$.

5 Advantatges and Disadvantatges

Using the Decision Tree algorithm has a certain number of advantages and disadvantages. The most important ones are listed below.

- Advantages:
  - This algorithm does not require much data pre-processing. So there is no need for normalization or scaling data. In addition, missing values in the context of tree construction are not affected either.
  - It can be used for classification and regression tasks.
  - This algorithm is non-parametric, meaning there are no assumptions about the spatial distribution and the classifier structure.
  - It is a very intuitive model and very easy to understand, visualize and explain.
  - It is a useful algorithm for data exploration. It is easy and fast to identify the most significant variables and relations between different variables.
  - It handles non-linear parameters efficiently
- Disadvantages:
  - The main problem of the Decision Trees is the overfitting.
  - If there is a small change in the data it can greatly affect the result obtained in the Decision Tree.
  - Sometimes the calculations in a Decision Tree agorithm can become very complex.
  - Sometimes it takes a long time to train the model using this algorithm.
  - It can not be used in big data.

6 Assumptions in our CART implementation

In our own implementation, these are the hyperparameters that are taken into account:

- Maximum depth (max_depth): The max_depth integer defines how deep should the tree grow. At the depth of max_depth, the searching for the best split feature and split feature values stops.
- Minimum samples split (min_samples_split): The min_samples_split integer defines the minimal number of observations in a node for the best split search to start. So for example, if the node has 30 observations but the min_samples_split = 35, then the growth of the tree stops.

Even though there exists more hyperparameters to regularize the complexity of the tree in the Sklearn implementation such as the number of nodes/leaves in the tree, in our implementation we only added two of them in order to control the growth of the tree. Some of these hyperparameters which are not implemented, are taken by default. For example, in Sklearn, the criterion hyperparameter, which is the function to measure the impurity, can be chosen among the Entropy and the GINI index. In our implementation is only considered the GINI index.

The main goal of this paper was to implement our CART version from scratch and compare the results with the one implemented in Sklearn, thus, all the input variables are treated as numeric variables. In this way, before growing the tree or training the DT, the features which are categorical are encoded with One-Hot-Encoding in order to transform them to numerical.

The implementation is done with 2 main function in order to grow the tree:

1. best_split: This function is calculating the GINI gain for each feature in the dataset and returns the one with the maximum gain and the value to split this feature.
2. grow_tree: recursive function that while the depth of the tree is less that depth_max and the number of samples is more than min_samples_split, calculates the best split if exists, splits the data and creates the left and right leaves for a particular node and repeats this process for each leaf.

7 Results

Here we have the results of the tree implemented by us printed. Here we can know for which feature we have done each of the splits, and with which threshold we have finally done it. For example, in the case of the sex variable, it can take values between 0 and 1, and we see that in the first split the chosen feature is Sex_female, when it is 0 it belongs to the left sub-tree, and when it is 1 it belongs to the right sub-tree. The same happens recursively for each sub-tree. In the case of the results below, we see that the next split in the right sub-tree is taking into account the feature "Fare" with the threshold value of 25.075.

```
Root
   | GINI impurity of the node: 0.48
   | Class distribution in the node: {0: 288, 1: 189}
   | Predicted class: 0
|-------- Split rule: Sex_female <= 0.5
           | GINI impurity of the node: 0.31
           | Class distribution in the node: {0: 246, 1: 58}
           | Predicted class: 0
|---------------- Split rule: Fare <= 25.075
                   | GINI impurity of the node: 0.19
                   | Class distribution in the node: {0: 181, 1: 22}
                   | Predicted class: 0
|------------------------ Split rule: Age <= 13.5
```

```
                          | GINI impurity of the node: 0.0
                          | Class distribution in the node: {1: 7}
                          | Predicted class: 1
|------------------------ Split rule: Age > 13.5
                          | GINI impurity of the node: 0.14
                          | Class distribution in the node: {0: 181, 1: 15}
                          | Predicted class: 0
|---------------- Split rule: Fare > 25.075
                  | GINI impurity of the node: 0.46
                  | Class distribution in the node: {0: 65, 1: 36}
                  | Predicted class: 0
|------------------------ Split rule: Age <= 53.0
                          | GINI impurity of the node: 0.48
                          | Class distribution in the node: {0: 47, 1: 33}
                          | Predicted class: 0
|------------------------ Split rule: Age > 53.0
                          | GINI impurity of the node: 0.24
                          | Class distribution in the node: {0: 18, 1: 3}
                          | Predicted class: 0
|-------- Split rule: Sex_female > 0.5
          | GINI impurity of the node: 0.37
          | Class distribution in the node: {1: 131, 0: 42}
          | Predicted class: 1
|---------------- Split rule: Fare <= 48.202
                  | GINI impurity of the node: 0.44
                  | Class distribution in the node: {1: 85, 0: 42}
                  | Predicted class: 1
|------------------------ Split rule: Fare <= 31.138
                          | GINI impurity of the node: 0.41
                          | Class distribution in the node: {0: 32, 1: 78}
                          | Predicted class: 1
|------------------------ Split rule: Fare > 31.138
                          | GINI impurity of the node: 0.48
                          | Class distribution in the node: {1: 7, 0: 10}
                          | Predicted class: 0
|---------------- Split rule: Fare > 48.202
                  | GINI impurity of the node: 0.0
                  | Class distribution in the node: {1: 46}
                  | Predicted class: 1
```

In the following table we have the metrics table obtained using our algorithm, with an accuracy of 76%.

```
              precision    recall  f1-score   support

           0       0.77      0.83      0.80       136
           1       0.74      0.66      0.70        99

    accuracy                           0.76       235
   macro avg       0.75      0.74      0.75       235
```

```
weighted avg       0.76      0.76      0.76       235
```

Finally, we also wanted to know the time it takes to generate the tree. We have seen that it takes 0.79 seconds to execute the algorithm implemented by us.

7.1 Comparing Results

We wanted to compare our results shown above with those obtained with the function DecisionTreeClassifier of the sklearn. For the comparison we have used the same two parameters used in our implementation: min_samples_split=50, max_depth=3.

Below we have printed the tree that is generated, where we can see that the features and splits are not exactly the same as the one above. In this case, the feature chosen for the first split is Sex_female, the same as in our implementation, but it is followed by the variable "Age", and here it is different from the previous results where it was the variable "Fare".

```
|--- Sex_female <= 0.50
|   |--- Age <= 6.50
|   |   |--- class: 1
|   |--- Age >  6.50
|   |   |--- Fare <= 26.27
|   |   |   |--- class: 0
|   |   |--- Fare >  26.27
|   |   |   |--- class: 0
|--- Sex_female >  0.50
|   |--- Fare <= 48.20
|   |   |--- Fare <= 10.48
|   |   |   |--- class: 1
|   |   |--- Fare >  10.48
|   |   |   |--- class: 1
|   |--- Fare >  48.20
|   |   |--- Age <= 8.00
|   |   |   |--- class: 0
|   |   |--- Age >  8.00
|   |   |   |--- class: 1
```

```
              precision    recall  f1-score   support

           0       0.80      0.82      0.81       136
           1       0.75      0.72      0.73        99

    accuracy                           0.78       235
   macro avg       0.77      0.77      0.77       235
weighted avg       0.78      0.78      0.78       235
```

As we can see in the table above, we have an accuracy of 78%, which is slightly higher than the one achieved with our algorithm, but the difference is still very small.

The execution time of this algorithm is 0.008 seconds. As we can see, in this case the time is much less than that obtained with what we have implemented, so the difference is relevant.

8 Conclusions

After all, and seeing the results achieved, we see that the accuracy obtained with our implementation is 76%, with a precision of 77% for class 0 and 74% for class 1, so it is a good result. If we compare these results with those obtained using sklearn, we see that the last ones are a little higher, since the accuracy is 78%, and the precision for class 0 is 80% and for class 1 is 75%. We can see that the difference obtained between both results is very small.

Moreover, another important thing to take into account is the time of execution required for each algorithm. The difference between the one implemented by us and the sklearn's is very different. This is something that we would have to improve for the next versions of this algorithm in such a way as to obtain a run time similar to the sklearn one.

This task was a good opportunity to learn how a decision tree works and learn what is behind it. These days exists a lot of APIs where it is used as a black box, thus, only with a dataset a classification task can be done easily. Therefore we learn how to grow a tree and how regularize its complexity after understanding hyperparameters. In this implementation, it is only taken into account the maximum depth of the tree and the minimum samples per tree. We arrived to the conclusion as our implementation is not configurable as it is the Sklearn version regarding the hyperparameters, the results, the final tree and classification metrics, are a slightly different. All in all, we learn in depth and understand the mathematics and the algorithm of the CART implementation. From now on, this paper can be continued with the implementation of the Random Forest from scratch based on this implementation.

9 Appendix

Here we leave the *best_split()* function where we show how given the features X and the target y the partition is calculated looking for the best feature with the best threshold for the split and comparing different splits and choosing the best one by calculating the GINI gain.

```
def best_split(self):

    df = self.X.copy()
    df['Y'] = self.Y
```

```
        GINI_base = self.get_GINI()
        max_gain = 0
        best_feature = None
        best_value = None

        for feature in self.features:
            Xdf = df.dropna().sort_values(feature)
            xmeans = self.ma(Xdf[feature].unique(), 2)

            for value in xmeans:
                left_counts = Counter(Xdf[Xdf[feature]<value]['Y'])
                right_counts = Counter(Xdf[Xdf[feature]>=value]['Y'])
                y0_left, y1_left, y0_right, y1_right = left_counts.get(0, 0), left_counts.get(1, 0), r
                gini_left = self.GINI_impurity(y0_left, y1_left)
                gini_right = self.GINI_impurity(y0_right, y1_right)
                n_left = y0_left + y1_left
                n_right = y0_right + y1_right
                w_left = n_left / (n_left + n_right)
                w_right = n_right / (n_left + n_right)
                wGINI = w_left * gini_left + w_right * gini_right
                GINIgain = GINI_base - wGINI

                if GINIgain > max_gain:
                    best_feature = feature
                    best_value = value
                    max_gain = GINIgain

        return (best_feature, best_value)
```

The function below, called *GINI_ impurity()* shows how the Gini Impurity value is calculated. This function is called in *best_ split()* to calculate the Gini value and decide the best split.

```
    def GINI_impurity(y1_count, y2_count):

        if y1_count is None:
            y1_count = 0
        if y2_count is None:
            y2_count = 0
        n = y1_count + y2_count
        if n == 0:
            return 0.0
        p1 = y1_count / n
        p2 = y2_count / n
        gini = 1 - (p1 ** 2 + p2 ** 2)
        return gini
```

In the function below we have the *grow_ tree()*. This function is called recursively to create the decision tree.

```
def grow_tree(self):

    df = self.X.copy()
    df['Y'] = self.Y

    if (self.depth < self.max_depth) and (self.n >= self.min_samples_split):

        best_feature, best_value = self.best_split()

        if best_feature is not None:
            self.best_feature = best_feature
            self.best_value = best_value

            left_df, right_df = df[df[best_feature]<=best_value].copy(), df[df[best_feature]>best_
            rule_left = f"{best_feature} <= {round(best_value, 3)}"
            rule_right = f"{best_feature} > {round(best_value, 3)}"
            left = Node(
                left_df['Y'].values.tolist(),
                left_df[self.features],
                depth=self.depth + 1,
                max_depth=self.max_depth,
                min_samples_split=self.min_samples_split,
                node_type='left_node',
                rule=rule_left
            )

            self.left = left
            self.left.grow_tree()

            right = Node(
                right_df['Y'].values.tolist(),
                right_df[self.features],
                depth=self.depth + 1,
                max_depth=self.max_depth,
                min_samples_split=self.min_samples_split,
                node_type='right_node',
                rule=rule_right
            )

            self.right = right
            self.right.grow_tree()
```

References

[Bha19] Saptashwa Bhattacharyya. 2019. URL: https://towardsdatascience.com/understanding-decision-tree-classification-with-scikit-learn-2ddf272731bd.

[R20] Arif R. 2020. URL: https://medium.com/analytics-vidhya/classification-in-decision-tree-a-step-by-step-cart-classification-and-regression-tree-8e5f5228b11e.

[Bir21] Himanshu Birla. 2021. URL: https://medium.com/analytics-vidhya/decision-tree-my-interpretation-part-i-e730aed60cd3.

[Buj21] Eligijus Bujokas. 2021. URL: https://towardsdatascience.com/decision-tree-algorithm-in-python-from-scratch-8c43f0e40173.

[Cha] Nagesh Singh Chauhan. URL: https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html#:~:text=Decision%5C%20trees%5C%20use%5C%20multiple%5C%20algorithms,respect.%5C%20to%5C%20the%5C%20target%5C%20variable.

[skl] sklearn. *1.10. decision trees.* URL: https://scikit-learn.org/stable/modules/tree.html.