Mehdi Mehdi

Ali Tawbi

Parallel programing

Decision Tree algorithm parallel version.

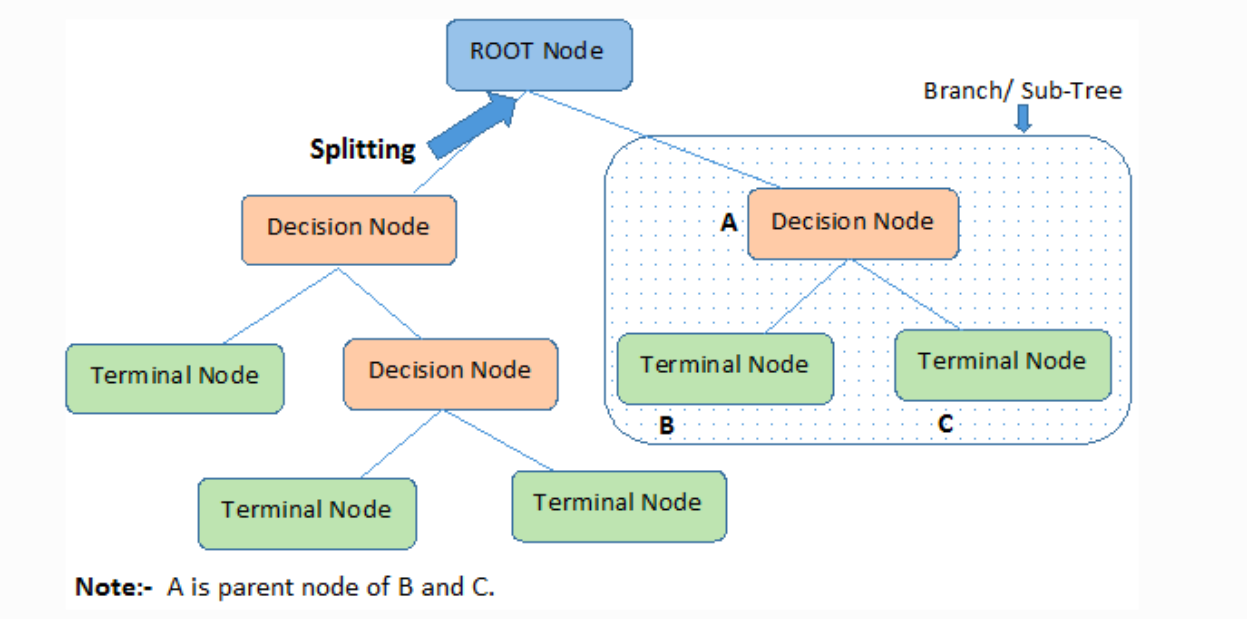
Introduction:

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving **regression and classification problems** too.

The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by **learning simple decision rules** inferred from prior data (training data).

Important Terminology related to Decision Trees:

1. **Root Node:** It represents the entire population or sample, and this further gets divided into two or more homogeneous sets.
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called the decision node.
4. **Leaf / Terminal Node:** Nodes do not split is called Leaf or Terminal node.
5. **Branch / Sub-Tree:** A subsection of the entire tree is called branch or sub-tree.
6. **Parent and Child Node:** A node, which is divided into sub-nodes is called a parent node of sub-nodes whereas sub-nodes are the child of a parent node.



How Decision Tree classification algorithm work

In Decision Trees, for predicting a class label for a record we start from the **root** of the tree. We compare the values of the root attribute with the record's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

Creating the Decision Tree

There are 2 important parameters where the algorithm uses them to create the tree:

1. min_samples : if the number of samples in a node less than this number make this node a terminal node.
2. max_depth: if the length of the tree become equal this number at a node make this node a terminal node.

Start with the root node:

If the number of samples < min_samples or the length of the tree is equal max_depth:

max_class = the class with the bigger number of samples.

make this node a terminal node with class max_class.

Else:

max_gain = - infinity, best_left =?, best_right =?.

best_threshold =?, best_feature =?

for every feature in features:

for every threshold in feature values:

left,right= split the samples with this threshold

gain = calculate information gain from this splitting

if gain > max_gain:

max_gain = gain, best_left = left, best_right = right

best_threshold =threshold, best_feature =feature.

Make this node Decision node with (best_threshold, best_feature)

Recursively apply the algorithm for the best_right and best_left.

To calculate the information gain we can use Gini index or Entropy.

If we have n features and m samples we need to check n x m threshold to find the best split.

Parallelize the algorithm:

To accelerate the finding best split we can assign the calculation for every threshold to a thread and then we choose the maximum gain.

Dataset:

Feature 1	Feature 2	Feature 3	…	…	…	Feature n
V1	V2	..	..	..	..	..
						Vnm

Feature 1	Feature 2	Feature 3	…	…	…	Feature n
Thread 1	Thread 2	…	…	…	…	Thread n
Thread n+1						
…						
…						
…						
…						
Thread (m-1)n +1						Thread mn

Assign every value(threshold) in the dataset to the to a thread, and this thread split the samples between left and right (if the value of the sample with same feature of the threshold larger than the threshold the sample goes right else left)

Then calculate the information gain of with this split.

We have in result a 2d array of information gains:

Gain 1	..	..	..	..	..	..
..	..	..	.	.	..	..
..	..	..	.	..	.	..
..	..	.		.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	Gain nm

After all the threads finished we get the max of this table (max_gain)

The index if the max_gain is the index of the best threshold to split the data.

Algorithm

The Red part in the previous code is the only part we need to change.

Create Threads more or equal the dataset size.

Result = copy of the dataset.

Thread number i:

    If i > size: die.

    Threshold = Dataset[i]

    Split and calculate gain.

    Result[i] = gain

Max = max(Result)

(best_threshold, best_feature) = index_of (Max, Dataset)

Left, right = split with (best_threshold, best_feature).

Result

```
classifier = DecisionTreeClassifier(min_samples_split=3, max_depth=3)
start_time = time.time()
classifier.fit(X_train,Y_train)
total_time = time.time()-start_time
print("Total time: ",total_time)
Y_pred = classifier.predict(X_test)
print("Accuracy: ",accuracy_score(Y_test, Y_pred))
```

```
Total time:  0.16599822044372559
Accuracy:  0.8666666666666667
```

```
classifier = DecisionTreeClassifierGPU(min_samples_split=3, max_depth=3)
start_time = time.time()
classifier.fit(X_train,Y_train)
total_time = time.time()-start_time
print("Total time: ",total_time)
Y_pred = classifier.predict(X_test)
print("Accuracy: ",accuracy_score(Y_test, Y_pred))
```

```
Total time:  0.029999494552612305
Accuracy:  0.8666666666666667
```

We apply the 2 algorithms on the same x_train and test it with the same x_test,y_test we had the same accuracy but the time of the parallel version is less than the sequential version.