Gradient Boosting Decision Trees - Doc

Introduction

The technique used is a supervised machine learning technique used mainly for regression problems.

First of all, gradient boosting is a technique to improve the predictive power of a weak learner. In our case, we used decision trees as additive models for gradient boosting. This is a sequential model in which after fitting model **`i`** we obtain some residuals which will be fitted by model **`i+1`**. The way to compute those residuals is:

$$\texttt{target\_value} - \sum_{j=1}^{i} \texttt{prediction\_by\_model[j]}$$

After obtaining the residual a new decision tree is used to fit the residuals. If one wants to predict the value for a new input the predictions for each estimator(decision tree) is summed up together. In this implementation, we did not use a very complex loss function in order to ease the implementation complexity.

The decision trees were also implemented. The learning algorithm used was ID3 for regression targets. So the best splitting decision was taken by maximizing the following formula(which is the same as saying that we want to maximize the reduction in variance when splitting the dataset that got to a given node) at a given node:

`dataset_size * dataset_variance - left_dataset_size * left_dataset_variance - right_dataset_size * right_dataset_variance`

Let's denote node **`i`** as the current node in the tree when learning the tree using the ID3 algorithm.
Where:
`dataset_size` is the size of the dataset that got to node **`i`**
`dataset_variance` is the variance in the target variable of the dataset at node **`i`**

Denote **`left_dataset`** as the dataset which would go to the left child of the current node after the split using a candidate split value.
`left_dataset_size` is the size of the **`left_dataset`**
`left_dataset_variance` is the variance of the target variable in the **`left_dataset`**

Denote **`right_dataset`** as the dataset which would go to the right child of the current node after the split using a candidate split value.
`right_dataset_size` is the size of the **`right_dataset`**
`right_dataset_variance` is the variance of the target variable in the **`right_dataset`**

During the ID3 algorithm when proposing candidate split points, if the number of unique elements in the feature is greater than 11, we only took the *percentiles* as split candidates, otherwise take *each* unique value in the feature as split candidate.

Use Cases

For gradient boosting, there are 2 use cases:

1. ***Regression***. The implementation is mainly oriented towards fitting an input that can have both continuous, ordinal and categorical features and whose target variable is a continuous one. It is worth mentioning that the ordinal and categorical features are treated the same way and the continuous feature must have the type **`numpy.float64.`**
2. ***Classification***. One can reduce the problem of classification to a problem of regression by converting the boolean/integer target variable into a continuous variable and predict those values, then using a threshold function.

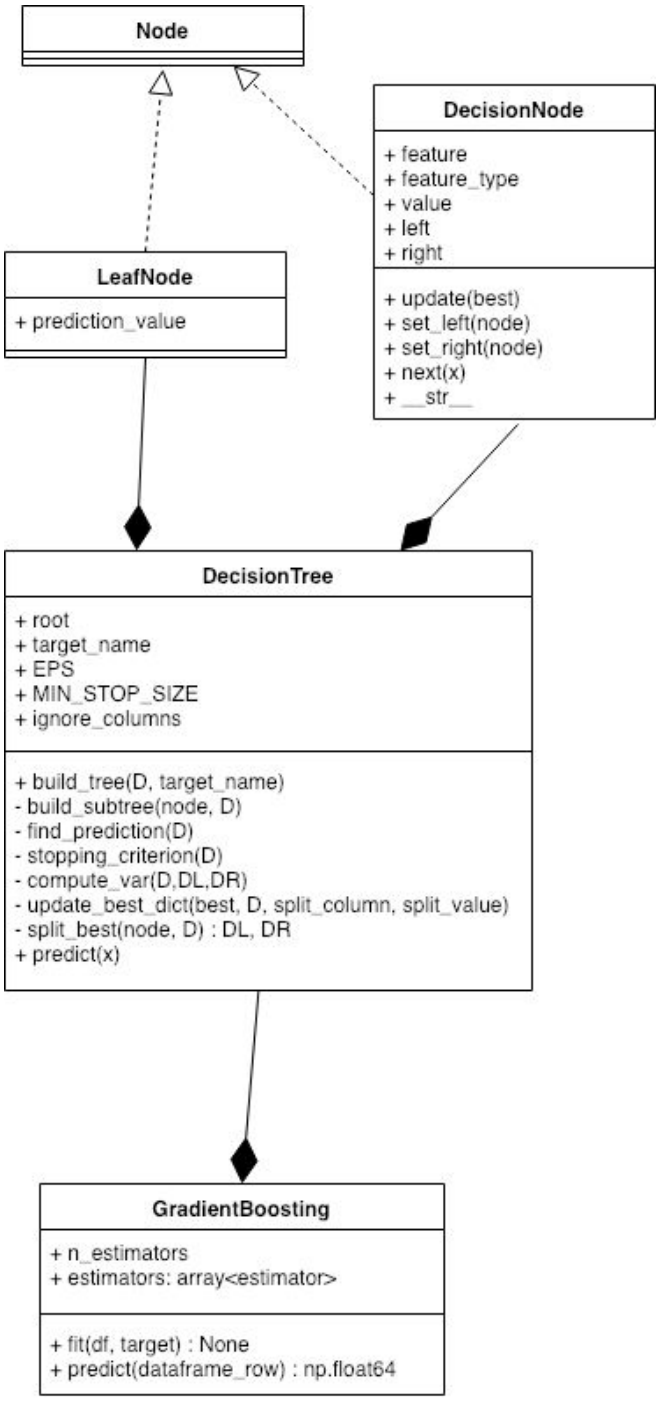
Usage

In order to **train** the model one must use **`.fit(df, target_name)`**, which fits the input represented by **`df`**. **`df`** must be a **`DataFrame`** object which contains the features of the input as objects and has a column that represents the target variable under the name **`target_name`**. If the target_variable is represented by an **integer** it must be converted to a **`np.float64`** type because the class accepts targets *only* of this form.

For **prediction** purposes one can use the **`.predict(x)`** method. **`x`** is a dataframe row having the features required to make the prediction.

In case one wants to use the decision tree as a model separately from the gradient boosting framework, one can use it in the following way: use **`.build_tree(df, target_name)`** as the **training** function and **`.predict(x)`** as a **prediction** function. The description of the arguments for both functions are the same as in the gradient boosting scenario.

Diagram


Node
DecisionNode
+ feature
+ feature_type
+ value
+ left
+ right
+ update(best)
+ set_left(node)
+ set_right(node)
+ next(x)
+ __str__
LeafNode
+ prediction_value
DecisionTree
+ root
+ target_name
+ EPS
+ MIN_STOP_SIZE
+ ignore_columns
+ build_tree(D, target_name)
- build_subtree(node, D)
- find_prediction(D)
- stopping_criterion(D)
- compute_var(D,DL,DR)
- update_best_dict(best, D, split_column, split_value)
- split_best(node, D) : DL, DR
+ predict(x)
GradientBoosting
+ n_estimators
+ estimators: array<estimator>
+ fit(df, target) : None
+ predict(dataframe_row) : np.float64