Light GBM Report

- *Donal Loitam*

INTRODUCTION

- LigthtGBM is a class of models called gradient boosters that uses tree based learning algorithms.
- Light GBM is prefixed as ‘Light’ because of its high speed.
- Light GBM can handle the large datasets and takes lower memory to run.
- Not just that, but CatBoost also offers accuracy just like the other algorithm in the tree family.
- The term CatBoost is an acronym that stands for "Category” and “Boosting.”
- CatBoost supports numerical, categorical, and text features but has a good handling technique for categorical data.

Features of LightGBM:

Optimization in Speed and Memory Usage

Many boosting tools use pre-sort-based algorithms (e.g. default algorithm in xgboost) for decision tree learning. It is a simple solution, but not easy to optimize.

LightGBM uses histogram-based algorithms, which bucket continuous feature (attribute) values into discrete bins. This speeds up training and reduces memory usage. Advantages of histogram-based algorithms include the following:

- ***Reduced cost of calculating the gain for each split***

  - Pre-sort-based algorithms have time complexity O(#data)
  - Computing the histogram has time complexity O(#data), but this involves only a fast sum-up operation. Once the histogram is constructed, a histogram-based algorithm has time complexity O(#bins), and #bins is far smaller than #data.

- ***Use histogram subtraction for further speedup***

  - To get one leaf's histograms in a binary tree, use the histogram subtraction of its parent and its neighbor
  - So it needs to construct histograms for only one leaf (with smaller #data than its neighbor). It then can get histograms of its neighbor by histogram subtraction with small cost (O(#bins))

- ***Reduce memory usage***

  - Replaces continuous values with discrete bins. If #bins is small, can use small data type, e.g. uint8_t, to store training data
  - No need to store additional information for pre-sorting feature values

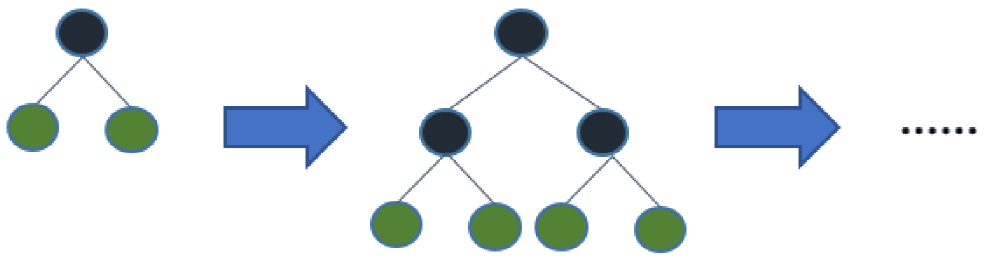
The Light GBM Algorithm :

<u>Leaf-wise (Best-first) Tree Growth</u>

Before we dive into the several differences that these algorithms possess, it should be noted that the CatBoost algorithm does not require the conversion of the data set to any specific format. Precisely numerical format, unlike XGBoost.

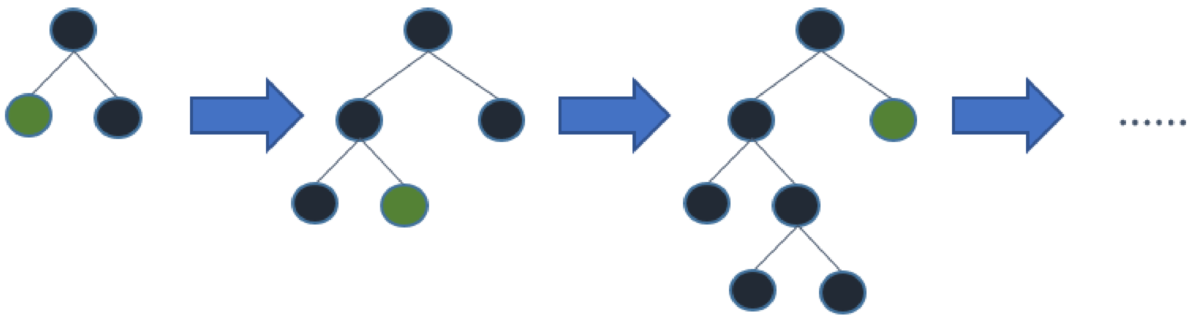
The algorithms differ from one another in implementing the boosted trees algorithm and their technical compatibilities and limitations.

Most decision tree learning algorithms grow trees by level (depth)-wise, like the following image:


Level-wise tree growth

LightGBM grows trees leaf-wise (best-first)[7]. It will choose the leaf with max delta loss to grow. Holding `#leaf` fixed, leaf-wise algorithms tend to achieve lower loss than level-wise algorithms.

Leaf-wise may cause over-fitting when `#data` is small, so LightGBM includes the `max_depth` parameter to limit tree depth. However, trees still grow leaf-wise even when `max_depth` is specified.


Leaf-wise tree growth

Categorical Features Handling

It is common to represent categorical features with one-hot encoding, but this approach is suboptimal for tree learners. Particularly for high-cardinality categorical features, a tree built on one-hot features tends to be unbalanced and needs to grow very deep to achieve good accuracy.

Instead of one-hot encoding, the optimal solution is to split on a categorical feature by partitioning its categories into 2 subsets. If the feature has `k` categories, there are `2^(k-1) - 1` possible partitions. But there is an efficient solution for regression trees[8]. It needs about `O(k * log(k))` to find the optimal partition.

The basic idea is to sort the categories according to the training objective at each split. More specifically, LightGBM sorts the histogram (for a categorical feature) according to its accumulated values (`sum_gradient / sum_hessian`) and then finds the best split on the sorted histogram.