Ensemble Learning is a technique that combines predictions from multiple models to get a prediction that would be more stable and generalize better. The idea is to average out different models' individual mistakes to reduce the risk of overfitting while maintaining strong prediction performance.

In regression, overall prediction is typically the mean of individual tree predictions, whereas, in classification, overall prediction is based on a weighted vote with probabilities averaged across all trees, and the class with the highest probability is the final predicted class.

There are two main classes of ensemble learning methods, namely bagging and boosting, although ML (Machine Learning) algorithms can be a combination of both with certain variations.

- **Bagging** method builds models in parallel using a random subset of data (sampling with replacement) and aggregates predictions of all models
- **Boosting** method builds models in sequence using the whole data, with each model improving on the previous model's error

CatBoost, LightGBM, and XGBoost are all variations of gradient boosting algorithms. Now you've understood the difference between bagging and boosting, we can move on to the differences in how the algorithms implement gradient boosting.

. . .

Catboost vs. LightGBM vs. XGBoost Characteristics

The table below is a summary of the differences between the three algorithms, read on for the elaboration of the characteristics.

	CatBoost	LightGBM	XGBoost
Developer	Yandex	Microsoft	DMLC
Release Year	2017	2016	2014
Tree Symmetry	Symmetric	Asymmetric Leaf-wise tree growth	Asymmetric Level-wise tree growth
Splitting Method	Greedy method	Gradient-based One-Side Sampling (GOSS)	Pre-sorted and histogram-based algorithm
Type of Boosting	Ordered	-	-
Numerical Columns	Support	Support	Support
Categorical Columns	Support Perform one-hot encoding (default) Transforming categorical to numerical columns by border, bucket, binarized target mean value, counter methods available	Support, but must use numerical columns Can interpret ordinal category	Supports, but must use numerical columns Cannot interpret ordinal category, users must convert to one-hot encoding, label encoding or mean encoding
Text Columns	Support Support Bag-of-Words, Naïve-Bayes or BM-25 to calculate numerical features from text data	Do not support	Do not support
Missing values	Handle missing value Interpret as NaN (default) Possible to interpret as error, or processed as minimum or maximum values	Handle missing value Interpret as NaN (default) or zero Assign missing values to side that reduces loss the most in each split	Handle missing value Interpret as NaN (tree booster) or zero (linear booster) Assign missing values to side that reduces loss the most in each split

Table 1: Characteristics of CatBoost, LightGBM, and XGBoost — Image by author

In CatBoost, symmetric trees, or balanced trees, refer to the splitting condition being consistent across all nodes at the same depth of the tree. LightGBM and XGBoost, on the other hand, results in asymmetric trees, meaning splitting condition for each node across the same depth can differ.

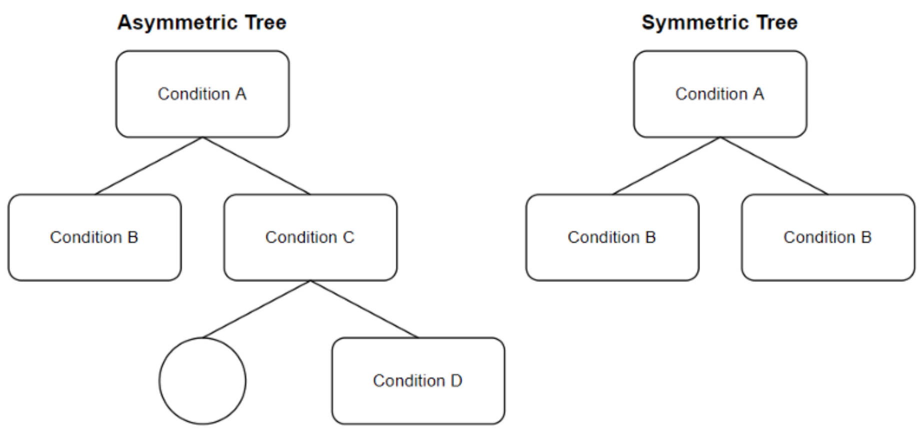

Fig 1: Asymmetric vs. Symmetric Trees — Image by author

For symmetric trees, this means that the splitting condition must result in the lowest loss across all nodes of the same depth. Benefits of balanced tree architecture include faster computation and evaluation and control overfitting.

Even though LightGBM and XGBoost are both asymmetric trees, LightGBM grows leaf-wise (horizontally) while XGBoost grows level-wise (vertically). To put it simply, we can think of LightGBM as growing the tree selectively, resulting in smaller and faster models compared to XGBoost.

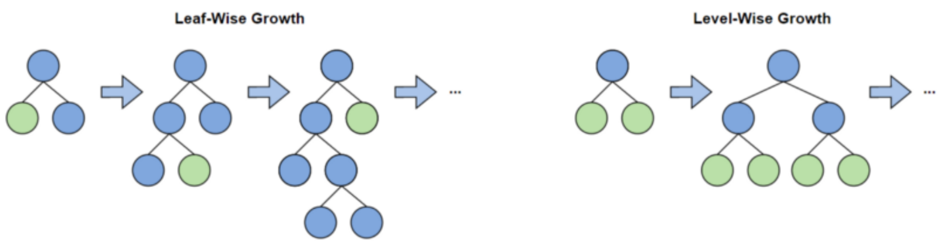

Fig 2: LightGBM (left) vs. XGBoost (right) — Image by author

Splitting Method

Splitting Method refers to how the splitting condition is determined.

In CatBoost, a greedy method is used such that a list of possible candidates of feature-split pairs are assigned to the leaf as the split and the split that results in the smallest penalty is selected.

the loss function. Data points with larger gradients have higher errors and would be important for finding the optimal split point, while data points with smaller gradients have smaller errors and would be important for keeping accuracy for learned decision trees. This sampling technique results in lesser data instances to train the model and hence faster training time.

In XGBoost, the pre-sorted algorithm considers all feature and sorts them by feature value. After which, a linear scan is done to decide the best split for the feature and feature value that results in the most information gain. The histogram-based algorithm works the same way but instead of considering all feature values, it groups feature values into discrete bins and finds the split point based on the discrete bins instead, which is more efficient than the pre-sorted algorithm although still slower than GOSS.

Type of Boosting

There are variations in how data is selected for training. Ordered boosting refers to the case when each model trains on a subset of data and evaluates another subset of data. Benefits of ordered boosting include increasing robustness to unseen data.

Categorical Columns

The parameters for categorical columns for different algorithms are as follows,

- CatBoost: `cat_features`, `one_hot_max_size`
- LightGBM: `categorical_feature`
- XGBoost: NA

. . .

Improving Accuracy, Speed, and Controlling Overfitting

In ensemble learning, averaging the prediction across different models helps with overfitting. However, as with any tree-based algorithm, there is still a possibility of overfitting. Overfitting can be handled in the splitting of the dataset into train, validation, and test set, enabling cross-validation, early stopping, or tree pruning. For the sake of comparing the different algorithms, we will focus on controlling overfitting using model parameters.

Note that to control the complexity of the model, XGBoost uses the parameter `max_depth` (since it grows level-wise) whereas LightGBM uses the parameter `num_leaves` (since it grows leaf-wise).

	CatBoost	LightGBM	XGBoost
Parameters to tune	*iterations*: number of trees *depth*: depth of tree *min_data_in_leaf*: control depth of tree	*num_leaves*: value should be less than 2^max_depth *min_data_in_leaf*: control depth of tree *max_depth*: depth of tree	*n_estimators*: number of trees *max_depth*: depth of tree *min_child_weight*: control depth of tree
Parameters for better accuracy		*max_bin*: maximum number of bins feature values will be bucketed in *num_leaves* Use bigger training data	
Parameters for faster speed	*subsample*: fraction of number of instances used in a tree *rsm*: random subspace method; fraction of number of features used in a split selection *iterations* *sampling_frequency*: frequency to sample weights and objects when building trees	*feature_fraction*: fraction of number of features used in a tree *bagging_fraction*: fraction of number of instances used in a tree *bagging_freq*: frequency for bagging *max_bin* *save_binary*: indicator to save dataset to binary file Use parallel learning	*colsample_bytree*: fraction of number of features used in a tree *subsample*: fraction of number of instances used in a tree *n_estimators*
Parameters to control overfitting	*early_stopping_rounds*: stop training after specified number of iterations since iteration with optimal metric value *od_type*: type of overfitting detector *learning_rate*: learning rate for reducing gradient step *depth* *l2_leaf_reg*: regularization parameter	*max_bin* *num_leaves* *max_depth* *bagging_fraction* *bagging_freq* *feature_fraction* *lambda_l1 / lambda_l2 / min_gain_to_split* Use bigger training data Regularization	*learning_rate* *gamma*: regularization parameter, higher gamma for more regularization *max_depth* *min_child_weight* *subsample*

Table 2: Parameters to tune for accuracy, speed, and overfitting — Image by author

. . .

Performance Comparison

There are various benchmarking on accuracy and speed performed on different datasets. I find it hasty to generalize algorithm performance over a few datasets, especially if overfitting and numerical/categorical variables are not properly accounted for.

However, generally, from the literature, XGBoost and LightGBM yield similar performance, with CatBoost and LightGBM performing much faster than XGBoost, especially for larger datasets.

. . .

Hope you have a better understanding of the three most popular types of ML boosting algorithms — CatBoost, LightGBM, and XGBoost which mainly differ structurally. In practice, data scientists usually try different types of ML algorithms against their data — so don't rule out any algorithm just yet! Besides understandability, performance, and timing considerations in choosing between different algorithms, it is also crucial to finetune the models via hyperparameter tuning and control overfitting via pipeline architecture or hyperparameters.

. . .

Related Links