Gradient Boosting

1) Algorithm export

Gradient Boosting is a machine learning technique for regression and classification problems. The prediction model generated by it is an integration of weak prediction models. For example, a typical decision tree is used as a weak prediction model. In this case, it is a gradient boosting tree (GBT or GBDT). Like other boosting methods, it builds models in a staged fashion, but it generalizes boosting methods in general by allowing optimization on arbitrary differentiable loss functions.

The Gradient Boosting algorithm iteratively combines multiple weak classifiers into a strong classifier. The simplest explanation is that in least squares regression, by minimizing the mean square error $\frac{1}{N}\sum_i(\hat{y}_i - y_i)^2$,Predict real values $\hat{y} = F(\mathbf{x})$

2) Training algorithm

Given a training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, The goal is to find a F ^ (x) that minimizes the expected value of some specified loss function L (y , F (x)) among all functions F (x) of a given form.

$$\hat{F} = \arg\min_F \mathbb{E}_{\mathbf{x},y}[L(y, F(\mathbf{x}))]$$

The gradient boosting method expresses F ^ (x) for estimating the real-valued variable y in the form of a weak learner (or base learner) h i (x)

in a certain class of H with weights.

$$\hat{F}(x) = \sum_{i=1}^{M} \gamma_i h_i(x) + \text{const.}$$

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma),$$

$$F_m(x) = F_{m-1}(x) + \arg\min_{h_m \in \mathcal{H}} \left[\sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \gamma_m h_m(x_i)) \right],$$

Unfortunately, selecting the best function h for an arbitrary loss function L at each step is usually a computationally infeasible optimization problem. Therefore, we restrict our approach to simplified versions of the problem.

The idea is to apply a gradient descent step to this minimization problem (function gradient descent). If we consider the continuous case, where H is the set R of arbitrary differential functions on, we update the model according to the equation

$$F_m(x) = F_{m-1}(x) - \gamma_m \sum_{i=1}^{N} \nabla_{F_{m-1}} L(y_i, F_{m-1}(x_i)),$$

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) - \gamma \nabla_{F_{m-1}} L(y_i, F_{m-1}(x_i))),$$

3. Several loss functions and corresponding Boosting algorithms

Different boosting algorithms correspond to different loss functions and

pseudo-residuals are summarized in the table below.

AdaBoost	index loss	$\exp\left(-yF(\mathbf{x})\right)$	$-y_i \exp\left(-y_i F_{m-1}\left(\mathbf{x}_i\right)\right)$
LS-Boost	mean square loss	$\frac{(y-F(\mathbf{x}))^2}{2}$	$y_i - F_{m-1}\left(\mathbf{x}_i\right)$
LAD-Boost	Absolute loss	$\lvert y-F(\mathbf{x})\rvert$	$\operatorname{sign}\left(y_i - F_{m-1}\left(\mathbf{x}_i\right)\right)$
M-Boost	Huber Loss	$\begin{cases} \frac{1}{2}(y-F(\mathbf{x}))^2, & \lvert y-F(\mathbf{x})\rvert \le \delta \\ \delta(\lvert y-F(\mathbf{x})\rvert - \delta/2), & \lvert y-F(\mathbf{x})\rvert > \delta \end{cases}$	$\begin{cases} y_i - F_{m-1}\left(\mathbf{x}_i\right), & \lvert y_i - F_{m-1}\left(\mathbf{x}_i\right)\rvert \le \delta \\ \delta \cdot \operatorname{sign}\left(y_i - F_{m-1}\left(\mathbf{x}_i\right)\right), & \lvert y_i - F_{m-1}\left(\mathbf{x}_i\right)\rvert > \delta \end{cases}$
LogitBoost (L_2-Boost)	Logistic Loss	$\log\left(1+e^{-yF(\mathbf{x})}\right)$	$\frac{y_i}{1+e^{y_i F_{m-1}(\mathbf{x}_i)}}$
L_K-Boost	多分类Logistic Loss	$-\sum_{k=1}^{K} y_k \log \frac{\exp\left(F_k(\mathbf{x})\right)}{\sum_{l=1}^{K}\exp\left(F_l(\mathbf{x})\right)}$	$\left\{ y_{ik} - \frac{\exp\left(F_{k,m-1}(\mathbf{x})\right)}{\sum_{l=1}^{K}\exp\left(F_{l,m-1}(\mathbf{x})\right)} \right\}_{k=1}^{K}$

XGBoost、Light GBM and CatBoost

1 Overview

In the case of deep learning, the boosting algorithm still has its place, especially when the training sample size is small, the training time is short, and there is no prior parameter adjustment, the boosting algorithm still maintains its advantages. In the kaggle competition, the boosting algorithm occupies the majority of seats. This article summarizes the introduction of the three algorithms in multiple documents and blogs, and compares these representative boosting algorithms from multiple perspectives to facilitate a deeper understanding.

2 Common points of the three algorithms

Structurally speaking, XGBoost, LightGBM, and CatBoost are all boosting algorithms, and their base learners are decision trees, and they all use greedy ideas to realize the growth of decision trees.

In practical applications, these algorithms supported by decision trees also integrate the good interpretability of decision trees. On some large data sets in Kaggle, these boosting algorithms can achieve quite good performance.

In terms of speed, LightGBM and CatBoost have made further optimization improvements on XGBoost, and the speed is generally faster than the previous XGBoost, and the parameter adjustment process is also much more convenient.

3 The difference between the three algorithms

3.1 Characteristics of trees

The three algorithm base learners are all decision trees, but there are still many differences in the characteristics of the trees and the process of generation

CatBoost uses a symmetric tree whose nodes can be mirrored. The tree model based on CatBoost is actually a complete binary tree.

The decision tree of XGBoost is Level-wise growth. Level-wise can split the leaves of the same layer at the same time, which is easy to perform multi-thread optimization, and the risk of overfitting is small, but this split method also has defects. Level-wise does not distinguish the leaves of the same layer, which brings a lot of unnecessary s expenses. In fact, the split gain of many leaves is low, and there is no need for searching and splitting.

The decision tree of LightGBM is Leaf-wise growth. Each time the leaf with the largest splitting gain (usually the one with the most data) is found from all the current leaves, the defect is that it is easy to grow a relatively deep decision tree and cause overfitting. In order to solve this problem, LightGBM uses Leaf -wise adds a limit on the maximum depth.

3.2 For categorical variables

When calling the boost model, when encountering a category variable, xgboost needs to process it first, and then input it into the model, while lightgbm can specify the name of the category variable, and it will be automatically processed during the training process.

Specifically, CatBoost can assign categorical variable indicators, and then obtain the result of the one-hot encoding form through the one-hot maximum (one-hot maximum: on all features, use the unique number for different numbers less than or equal to a given parameter value one-hot encoding; also, set "skip" in the CatBoost statement, and CatBoost will treat all columns as numeric variables).

LightGBM can also handle attribute data by using input of feature names; it does not one-hot encode the data, so it is much faster than one-hot

encoding. LGBM uses a special algorithm to determine the split value of attribute features. (Note: The categorical variable needs to be converted to an integer variable; this algorithm does not allow string data to be passed to the categorical variable parameter)

Unlike CatBoost and LGBM algorithms, XGBoost itself cannot handle categorical variables and only accepts numerical data, which is very similar to RF. In actual use, before the classification data is passed into XGBoost, the data must be processed by various encoding methods such as label encoding, mean encoding or one-hot encoding.

3.3 Breakthrough of XGBoost

XGBoost can be regarded as a breakthrough in the development of the Boosting algorithm. First, it uses the second-order gradient to divide the nodes. Compared with other GBMs, the accuracy is higher; second, using the local approximation algorithm to optimize the greedy algorithm of the split node, we can Adjust and select the appropriate eps to maintain the performance of the algorithm and improve the operation speed of the algorithm; in addition, XGBoost also adds L1/L2 items to the loss function to control the complexity of the model and improve the stability of the model; providing parallel computing capabilities is also XGBoost a feature.

3.4 Optimization made by LightGBM

Some optimizations made by LightGBM on the algorithm are worthy of attention. First, LightGBM uses a histogram based algorithm to bin and discretize continuous features. The advantage of this is that it can improve training speed and save storage space. A histogram-based algorithm replaces the data structure previously built by XGBoost with pre-sorted. (So the operation of LGBM is O(bins) instead of O(data). Of course, O(data) is still required in the process of binning (only sum operation))

In addition, LightGBM uses GOSS Gradient-based One-Side Sampling (GOSS Gradient-based One-Side Sampling), which maintains samples with high gradients and performs random sampling from samples with small gradient changes.

4 Supplements on parameters

4.1 Related to categorical variables

Parameters related to category variables are usually adjusted in CatBoost and LightGBM (because XGBoost does not have such parameters), cat_features is a parameter for feature indexing in CatBoost. In addition, CatBoost can use one_hot_max_size to perform one-hot encoding for all features. The value of this parameter is the upper limit of the number of features after one-hot

encoding (less than 255).

In LightGBM, the categorical_feature parameter is generally used to select the categorical features we want to use during the training process. This parameter helps us quickly build a suitable training model.

4.2 Speed Control Parameters

Such parameters of the three boosting methods are different. In XGBoost, use colsample_bytree to control the subsampling rate of column sampling, use subsample to control the subsampling rate of training data, and use n_estimators to control the maximum number of decision trees; in CatBoost, use rsm to select the method of random subsampling, That is, the percentage of the number of features used in each split, and the maximum number of decision trees are controlled by iterations; in LightGBM, feature_fraction is used to control the part of the feature that can be used in each iteration, and bagging_fraction is used to control the data used in each iteration (favorable to improve Training speed), use num_iterations to control the number of boosting iterations.

First let's know how the preclassification algorithm works:

1. For each node, iterate over all features
2. For each feature, classify the examples according to the feature value
3. Perform a linear scan to determine the optimal segmentation based on the
4. basic information gain of the current feature
5. Select the best of all feature segmentation results

When filtering data samples to find segmentation values, LightGBM uses a new technique: gradient-based one-sided sampling (GOSS); while XGBoost uses pre-classification algorithms and histogram algorithms to determine optimal segmentation. An instance here means an observation/sample.

How does each model handle attribute categorical variables?

CatBoost can assign categorical variable indicators, and then obtain one-hot encoding results through one-hot maximum (one-hot maximum: on all features, use one-hot encoding for different numbers less than or equal to a given parameter value).

If "skip" is not set in the CatBoost statement, CatBoost will treat all columns as numeric variables.

Note that if a column of data contains string values, the CatBoost algorithm

will throw an error. In addition, int type variables with default values will also be treated as numeric data by default. In CatBoost, variables must be declared in order for the algorithm to treat them as categorical variables.

> Note: If a column having string values is not provided in the cat_features, CatBoost throws an error. Also, a column having default int type will be treated as numeric by default, one has to specify it in cat_features to make the algorithm treat it as categorical.

```
from catboost import CatBoostRegressor
# Initialize data
cat_features = [0,1,2]
train_data = [["a","b",1,4,5,6],["a","b",4,5,6,7],["c","d",30,40,50,60]]
test_data = [["a","b",2,4,6,8],["a","d",1,4,50,60]]
train_labels = [10,20,30]
# Initialize CatBoostRegressor
model = CatBoostRegressor(iterations=2, learning_rate=1, depth=2)
# Fit model
model.fit(train_data, train_labels, cat_features)
```

For categorical variables whose number of possible values is larger than the one-hot maximum, CatBoost uses a very effective encoding method, which is similar to mean encoding, but can reduce overfitting.

LightGBM

Like CatBoost, LightGBM can also process attribute data by using the input of feature names; it does not one-hot encode the data, so it is much faster than

one-hot encoding. LGBM uses a special algorithm to determine the split value of attribute features.

Specific feature names and categorical features:

```
train_data = lgb.Dataset(data, label=label, feature_name=['c1', 'c2', 'c3'],␣
↪categorical_feature=['c3'])
```

It is important to know that categorical variables need to be converted to integer variables before building a dataset suitable for LGBM; this algorithm does not allow string data to be passed to the categorical variable parameter.

LightGBM

Like CatBoost, LightGBM can also process attribute data by using the input of feature names; it does not one-hot encode the data, so it is much faster than one-hot encoding. LGBM uses a special algorithm to determine the split value of attribute features.

Function	XGBoost	CatBoost	Light GBM
Important parameters which control overfitting	1. **learning_rate or eta** – optimal values lie between 0.01-0.2 2. **max_depth** 3. **min_child_weight:** similar to min_child leaf; default is 1	1. **Learning_rate** 2. **Depth** - value can be any integer up to 16. Recommended - [1 to 10] 3. No such feature like min_child_weight 4. **l2-leaf-reg**: L2 regularization coefficient. Used for leaf value calculation (any positive integer allowed)	1. **learning_rate** 2. **max_depth**: default is 20. Important to note that tree still grows leaf-wise. Hence it is important to tune **num_leaves** (number of leaves in a tree) which should be smaller than 2^(max_depth). It is a very important parameter for LGBM 3. **min_data_in_leaf**: default=20, alias= min_data, min_child_samples
Parameters for categorical values	Not Available	1. **cat_features**: It denotes the index of categorical features 2. **one_hot_max_size**: Use one-hot encoding for all features with number of different values less than or equal to the given parameter value (max – 255)	1. **categorical_feature:** specify the categorical features we want to use for training our model
Parameters for controlling speed	1. **colsample_bytree**: subsample ratio of columns 2. **subsample**: subsample ratio of the training instance 3. **n_estimators**: maximum number of decision trees; high value can lead to overfitting	1. **rsm:** Random subspace method. The percentage of features to use at each split selection 2. No such parameter to subset data 3. **iterations**: maximum number of trees that can be built; high value can lead to overfitting	1. **feature_fraction:** fraction of features to be taken for each iteration 2. **bagging_fraction:** data to be used for each iteration and is generally used to speed up the training and avoid overfitting 3. **num_iterations:** number of boosting iterations to be performed; default=100

https://en.wikipedia.org/wiki/Gradient_boosting
https://en.wikipedia.org/wiki/XGBoost
https://en.wikipedia.org/wiki/LightGBM
https://en.wikipedia.org/wiki/CatBoost
https://towardsdatascience.com/catboost-vs-lightgbm-vs-xgboost-c80f40662924
https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db
https://cloud.tencent.com/developer/article/1814287