Hindawi
Security and Communication Networks
Volume 2022, Article ID 5156086, 8 pages
https://doi.org/10.1155/2022/5156086

Research Article

LGBM-CBFS: A Heuristic Feature Sampling Method Based on Tree Ensembles

Yu Zhou, Hui Li, and Mei Chen

College of Computer Science and Technology, Guizhou University, Guiyang, China

Correspondence should be addressed to Hui Li; cse.huili@gzu.edu.cn

Received 26 November 2021; Accepted 14 February 2022; Published 16 March 2022

Academic Editor: Chenquan Gan

Gradient boosting decision tree (GBDT) is widely used because of its state-of-art performance in academia, industry, and data science competitions. The efficiency of the model is limited by the overwhelming training cost with the surge of data. A common solution is data reduction by sampling on training data. Current popular implementations of GBDT such as XGBoost and LightGBM both supports cut the search space by using only a random subset of features without any prior knowledge, which is ineffective and may lead the model fail to converge when sampling on a high-dimensional feature space with a small sampling rate assigned. To mitigate this problem, we proposed a heuristic sampling algorithm LGBM-CBFS, which samples features based on an available prior knowledge named "importance scores" to improve the performance and the effectiveness of GBDT. Experimental results indicate that LGBM-CBFS obtains a higher level of model accuracy than uniform sampling without introducing unacceptable time cost in the sparse high-dimensional scenarios.

1. Introduction

Gradient boosting decision tree [1] achieves state-of-the-art performance in machine learning and data mining applications ranging from classification, regression to ranking. The rapid growth of data volume leads to expensive training cost. The major computation cost of training the GBDT model comes from building a decision tree in each gradient boosting round, in which finding the optimal split point results in the most time-consuming operation because it requires scanning entire training data for each feature. Recently, two open-source projects XGBoost [2] and LightGBM [3] have been widely used for their superiority in machine learning and data analytic applications, which exploit histogram-based data partitioning to accelerate decision tree building. However, both of them still suffer from performance degradation in large-scale learning tasks at a limited time and economic cost.

To alleviate this problem, sampling techniques are commonly used to reduce the data size by obtaining a random subset from the input dataset with both row-oriented and column-oriented perspectives. Current sampling research includes static sampling and adaptive sampling techniques. The former one draws samples based on a preset sample size determined prior to the start of sampling, which encounters the difficulty of determining an appropriate sample size. Chernoff-Hoeffding bounds [4, 5] have been widely used to test whether the number of samples seen so far is sufficient to assure estimation accuracy and confidence. Nevertheless, a static sampling scheme tends to overestimate the sample size due to the worst situation considered at the beginning of sampling, which is rare in practical applications [6]. Adaptive sampling [7–9] is proposed to address this situation by picking samples in an online fashion, and its stopping condition depends on the number of observations, namely, random samples seen so far. There are some studies [10, 11] that seek a tighter bound on sufficient sample size without jeopardizing accuracy and confidence. Compared to static sampling, adaptive sampling can approximate input datasets well with a lower sample size which achieves better scalability of learning algorithms. Nevertheless, sampling randomly on a high-dimensional dataset with a quite small sample size often makes the model hard to converge in an adaptive sampling scheme since most features involved in the training set are sparse.

Currently, the representative open-source implementations of GBDT such as XGBoost and LightGBM used uniform sampling to choose a subset of features. It is a simple and effective way to reduce the dimensions of feature space with a low time consumption. Nevertheless, there are no prior insights taken into account in the sampling process in this sampling scheme. Uniform sampling with a small sample size tends to be underperforming in sparse scenarios since there is little information that can be learned. Scores-based nonuniform sampling [12, 13] will obtain further performance improvement by picking a representative subset of features according to a metric that indicates the importance of the features. However, it will bring additional time costs due to the calculation of scores introduced.

To address the aforementioned issues, in this paper, we devised a sampling method named LGBM-GBFS to choose features based on an available prior "importance score," which reveals the contribution of every single feature to information gain in GBDT, meanwhile, and heavily reduced the computational cost bringing by building scores. Then, the features with high scores will be kept in the training set. The features with low scores will be discarded at random with a certain probability to cut the feature space. We integrate our LGBM-GBFS algorithm into LightGBM and conduct a series of experiments to verify the correctness and effectiveness of the approach. The experimental results show that our algorithm significantly improved the effectiveness when sampling features on high-dimensional sparse data, and it works better than uniform sampling without additional heavy computational cost.

2. Related Work

In this section, we present a brief review of gradient boosting and elaborate the related work of feature sampling for ensemble learning.

2.1. Review of GBDT. Gradient boosting tree is an ensemble learning method that takes decision tree as its base learner. It is proposed first by Friedman in [1] and is increasingly adopted as a crucial solution in data analytics with the advent of more efficient implementations. XGBoost and LightGBM are two well-known GBDT implementations that achieve outstanding performance in learning applications like ranking as well as classification. Both of them exploit histogram-based algorithms to find the optimal split point and employ many engineering or theoretical optimizations to improve the efficiency and scalability. For instance, XGBoost uses the second-order Taylor expansion as an approximation for the objective function in the general setting and introduces a well-designed regularization term to make the objective easy to parallelize.

In XGBoost, for a given dataset with n instances and m features $\mathrm{D} = \{(x_i, y_i)\}_1^n (x_i \in \mathbb{R}^m, y_i \in \mathbb{R})$, the general regularized objective is given by

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_k \Omega(f_k)\ ,\ f_k \in \mathrm{F} \tag{1}$$

where $\Omega(f) = \gamma T + (1/2)\lambda \sum_{j=1}^{T} \omega_j^2$ is the regularization term, in which T and ω are the number of leaves and leaf weights of one single decision tree. l is the conventional loss function for tree boosting. F is the functional space of regression trees. k is the number of decision trees we build. The tree boosting method in XGBoost rewrites the loss function by applying forward stagewise additive modeling. Then take Taylor expansion of the objective which is given by $Obj^{(t)} = \sum_{j=1}^{T} [G_j \omega_j + (1/2)(H_j + \lambda)\omega_j^2] + \gamma T$, where $G_j = \sum_{i \in I_j} g_i$, $H_j = \sum_{i \in I_j} h_i$. Here, I_j is the instance set in leaf j. g_i and h_i are first- and second-order gradient statistics on l. With this, the optimal j-th leaf weight and objective function value can be calculated by

$$\omega_j^* = -\frac{G_j}{H_j + \lambda}, Obj^* = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T. \tag{2}$$

The loss reduction of objective for each split is given by the following equation:

$$Gain = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma, \tag{3}$$

where G_L and G_R are the sum of the first-order gradient statistics of the instances on the left and right nodes after splitting. H_L and H_R are the sum of the second-order gradient statistics after splitting. The regularization coefficient γ indicates the complexity cost of additional leaf nodes added.

The feature importance score is obtained by (3) when taking the best split solution, which is used as a measure of the importance of a feature.

In order to conduct further optimization based on the work of XGBoost, LightGBM proposed GOSS (gradient-based one-side sampling) to choose a subset of the instances by keeping those instances with gradients greater than a preset threshold and sampling on the rest of the instances randomly. GOSS works better than uniform sampling from an accuracy perspective since more under-trained instances with a potentially greater contribution are involved in training data. Moreover, LightGBM proposed EFB (exclusive feature bundling) to convert data with quite sparse feature space into a dense form by bundling those features rarely taking nonzero values simultaneously together. Both GOSS and EFB give over 20x speedup of the conventional training process and achieve nearly the same accuracy.

To take advantage of the efficiency of LightGBM, we extend it to support the proposal sampling algorithm in this paper and conduct experiments based on the modification version. More details of the modifications are introduced in section 3.2.

2.2. Sampling Schemes in Ensemble Learning. Sampling is an effective technique widely used to improve the generalization and scalability in large-scale machine learning. Many researches related to sampling in ensemble learning are proposed in recent years. SGB (stochastic gradient boosting) [14] trains the base learner on a subset sampled uniformly from input data in a static way, which requires the sample

ratio is known in advance. This approach is also supported by XGboost and LightGBM. Watanabe proposed an adaptive sampling scheme MadaBoost [6] to dynamically adjust sample size in the training progress by using theoretical bounds like Chernoff-Hoeffding bound [4, 15]. MadaBoost achieves a higher accuracy with data of the same size as the static sampling method. Inspired by the works in MadaBoost, a new adaptive sampling scheme [10] proposed by Jianhua Chen et al. recently achieves a tighter bound of sample size by using Massart's inequality. Empirically, feature sampling outperforms row sampling in preventing overfitting. It is as well as used to reduce the search space in tree ensembles to improve training speed when finding the best split. Off-the-shelf feature sampling schemes fall into four general categories: uniform sampling, reweight-based sampling, score-based sampling, and bandit-based sampling. Uniform sampling is the simplest one that chooses columns with equal probability with taking no prior into account. It is the default and unique column sampling strategy in XGBoost, LightGBM, RandomForest [16]. [17, 18]. Reweight-based sampling first trains base learners on a random subset of the data instances in the first iteration and then chooses columns in terms of the suitable weight like classification error [17] in the previous iteration. The score-based strategy [19–22] introduces an importance sampling distribution to sample the features. Bandit-based sampling chooses columns by making a tradeoff between those with large past rewards and those never selected in every boosting round.

Uniform sampling has been widely used due to its simplicity and low computational cost. But it accepts important features with quite a low probability in high-dimension feature space. Moreover, it comes with a penalty in prediction accuracy due to taking no prior information into account. The latter three sampling schemes all construct an "importance score" for each feature of the input dataset to measure the impact of the corresponding feature on the model output. Then perform biased random sampling based on the scores in previous step. Schemes (2), (3), and (4) can improve the test accuracy effectively but require more computation cost due to the extra step of score construction.

In this paper, instead of constructing a new importance score, we select features by using the gain information of GBDT formulated by (3) to avoid additional calculation. We propose a column subsampling algorithm LGBM-CBFS, which chooses columns by filtering out those over-trained to obtain better efficiency. And we implement it into LightGBM with a minor modification. The extensive experimental results show our sampling algorithm obtained better accuracy than uniform sampling with acceptable overheads.

3. Contribution-Based Feature Sampling Algorithm

In this section, we train GBDT models on various datasets with different distributions and sparsity to study their differences with importance scores. And then, we implement our contribution-based feature sampling algorithm named LGBM-CBFS based on our analysis.

3.1. Importance Scores in High-Dimensional Data. Importance scores of each feature in GBDT indicate how much a feature contributes to the model output. In order to observe the difference in scores among the features of the training set, we build GBDT models on four publicly available data sets (real-sim, news20, kdda, and higgs) and details are described in Table 1, and we use a pie chart to depict the final contribution of each feature in an intuitive way. Features are automatically named in terms of their index in the input array of the charts. The GBDT model calculates importance scores by first adding up the value a feature split point improving the performance measure for a decision tree, then averaging across all the trees in the model. The more a feature is important, the higher it scores. Figure 1 shows the ratio of different feature scores to the total, which is sorted in descending order before plotting. We can see a few features contribute a lot at the overall level in real-sim, news20, and kdda, and there are relatively small differences in higgs.

We classify features into two categories that refer to as key features and valid features according to the effect on model output. Valid features represent features whose importance score is greater than 0. Key features represent features ranking in 80% of scores. The number of features, valid features, and key features is shown in Table 2. In the case of News20, the dataset consists of 50K features, of which 1090 features have scores greater than 0. And 79 of the 1090 features provide 80% of the total importance scores. Moreover, the number of valid features accounted for 2.18%, 3.8%, and 0.0058% of the total number of features in news20, real-sim, and kdda, respectively. The number of key features accounted for about 16.4%, 7.65%, and 17% of the number of valid features in news20, real-sim, and kdda, respectively. It can be seen that valid features have a lower proportion in high-dimensional sparse data, which means a low sampling probability of key features.

We plot the cumulative contributions of key features for every dataset in Figure 2 to show the change of importance scores at each boosting round. From Figure 2, we can observe that the ranking of certain features varies little during the training process. It means the importance of features from the current iteration to the next iteration is relatively stable. Besides, we can conclude that the features with high scores in the current iteration may have more impact on model output in the next iteration.

Here, we assume that features are independent of each other. When training the GBDT models on a high-dimensional sparse data set, the characteristics of importance scores could be summarized as follows:

(1) There is a considerable difference among the features in sparse data. Only a few features (valid features) play an important role in contributing to the information gain, in which the proportion of key features, that contribute most of the overall level, is quite small.

Table 1: Experimental datasets.

Dataset	#Data	#Features	Description
Gamma Norm Poisson Uniform	100K	500	Dense
Higgs	10.5 M	28	Sparse
News20	6K	5K	
Real-sim	72309	20958	
Kdda	8407752	20216830	

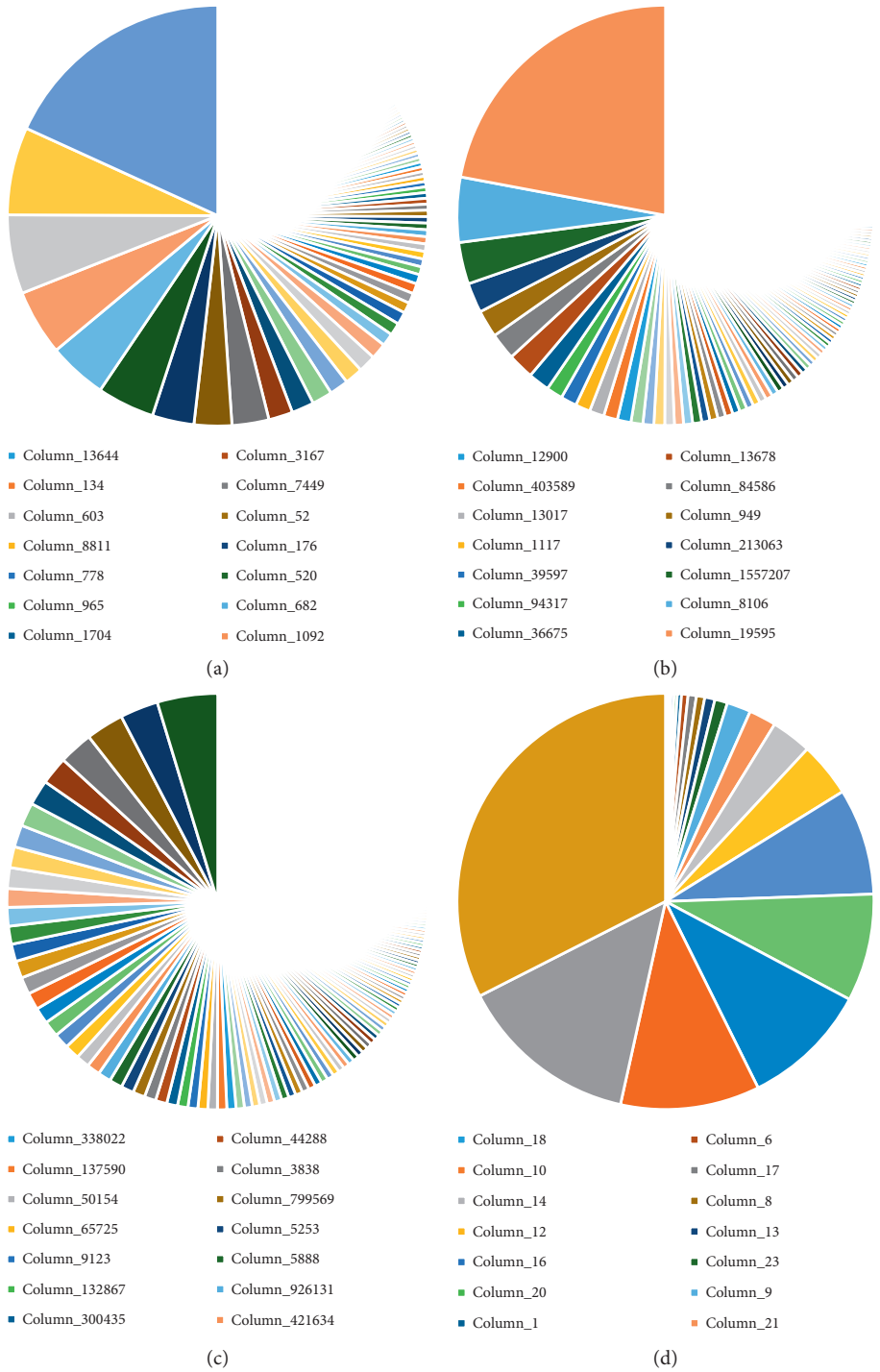

Figure 1: The proportion of importance scores (gain). (a) Real-sim. (b) News20. (c) kdda. (d) higgs.

Table 2: The number of different types of features.

Dataset	#Features	#valid_features	#key_features
Higgs	28	22	6
News20	50K	1090	179
Real-sim	20958	797	61
Kdda	20216830	1170	199

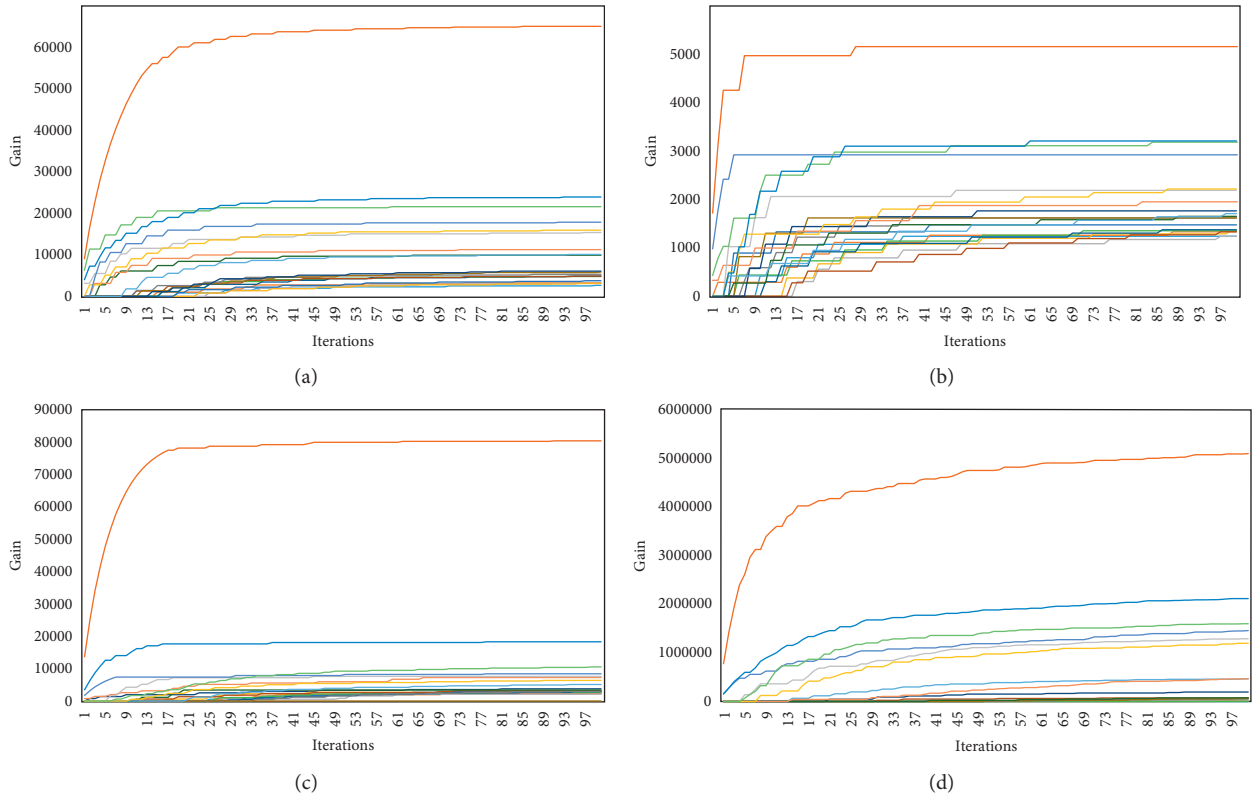

Figure 2: Accumulated values of importance score in the training process. (a) Real-sim. (b) News20. (c) higgs. (d) kdda.

(2) There are little differences in cumulative contribution to the information gain of each feature, but the contribution of a particular feature in a single iteration generally decreases as the number of iterations increases.

(3) In GBDT, key features are sampled at random with the same quite small probability, when high-dimensional sparse data are fed into the training program and a small sampling rate is given. That is why uniform sampling does not work in this case.

In summary, we can choose features based on the cumulative contribution, which is the importance scores of the features at one single iteration. It improves the effectiveness of the training process by keeping the key features are sampled into the training set.

3.2. LGBM-CBFS: LightGBM with Contribution-Based Feature Sampling. In this subsection, we devised a contribution-based feature sampling algorithm named LGBM-CBFS and integrate it into LightGBM. The details of the algorithm and the modifications made to LightGBM will be described.

The pseudocode of LGBM-CBFS is illustrated in Algorithm 1. Three inputs are fed into the function selectTopN. The first parameter $featImportances$ denotes the importance scores of valid features at one boosting iteration. The second parameter *vali dF eatIn di ces* represents the position indices of valid features, which has the same size with $featImportances$. The last parameter $featCount$ describes the number of features added to the final training set. At the beginning of the algorithm, we sort the features based on importance scores in descending order (line 3). Then, calculating the number of features with high scores needs to be retained in the result set *i dx Selecte d* (line 4, 5). The built-in parameter $feature\_fractoin$ indicates the feature sampling rate. p is the value of the parameter $reamin\_feature\_ratio$ that indicates the proportion of high-score features we keep. Here, we take an empirical value of 0.5 for $reamin\_feature\_ratio$. Not only possible significant features but also randomness is kept during the sampling process. After this process, according to the value obtained

```
Algorithm SelectTopN
Input: featImportances//the vector containing importance scores of valid features
Input: validFeatIndices//the vector containing position indices of valid features
Input: featCount//the number of valid features
(1)  i dx Selecte d← //the result set
(2)  //sort features based on scores
(3)  sorte dI dx← Sort(validFeatIndices, featImportances)
(4)  importantFeatNum← validFeatIndices.size()*feature_fraction*, p
(5)  importantFeatNum← Min(importantFeatNum, featCount)
(6)  i dx Selecte d← The first importantFeatNum elements in sorte dI dx
(7)  samplingNum←featCount − nonRan dC nt
(8)  ran dI dx←rand.get(sorte dI dx − i dx Selecte d, samplingNum)
(9)  i dx Selecte d←i dx Selecte d⋃ samplingNum
(10) Return i dx Selecte d
```

Algorithm 1: Contribution-based feature sampling algorithm.

in the previous step, the corresponding feature indices are included in result set (line 6). Finally, we sample uniformly from the rest of the features and merge them into *i dx Selecte d* as a final result (line 7,8,9).

To make our algorithm is easy to use, we make some necessary modifications to LightGBM. Two parameters *feature_sampling_metho d* and *remain_feature_ratio* are added to source file config.h which is defined in the *include* module and contains descriptions of all parameters in LightGBM. Our sampling algorithm is enabled when *feature_sampling_method* set as LGBM-CBFS. User can determine how many features with high scores will be kept by setting the value of *remain_feature_ratio*. We define an interface for LGBM-CBFS by implement it in a new source file feature_sampling.cpp of boosting module. The calling code is added before building a decision tree at one iteration, which is added to gbdt.cpp. Now that only a few lines in LightGBM were modified to provide an accessible interface to users.

4. Experiments

In this section, we compare LGBM-CBFS with the built-in feature sampling algorithm in LightGBM from an accuracy and speed perspective. The datasets used in our experiments comprise four artificial datasets generated by MATLAB and four publicly available datasets. The number of data instances and features are shown in Table 1. All experiments run on a single computer with two quad core i7-4790 CPUs and 16G memories.

4.1. Accuracy Evaluation. All of the artificial datasets in our experiments consist of 500 features and 100K instances exported by a 100K-by-500 matrix in MATLAB. We generate dataset Gamma from the gamma distribution with shape parameter in 2 and scale parameter in 500. Dataset Norm is generated from the normal distribution with mean parameter in 0 and deviation parameter in 500. We generate Poisson datasets from the Poisson distribution specified by the rate parameter $\lambda = 2$. A 100K-by-500 matrix of uniformly distributed random numbers is generated between 0 and 1000. Our experimental datasets cover different distribution and sparsity and list in Table 1 in detail.

To verify the correctness and effectiveness of LGBM-CBFS, we train three GBDT models, respectively, by specifying different sampling strategies on real-sim and kdda datasets and recording their binary error and AUC as performance metrics at every boosting iteration. Our experimental results are depicted in Figure 3, in which we use LGBM, and the orange line denotes the performance of the model trained with all features. LGBM_FS and the green line denote the performance of the model built on features sampling by built-in strategy in LightGBM, which is picking a subset of features uniformly at random. LGBM-CBFS and the pink line denote the performance of the model trained on features sampling by using the proposed approach in every boosting round.

In our experiments, a small sampling rate is assigned to evaluate the performance of our algorithm. We set the sampling rate to 0.05% in our experiment. The results in Figure 3 can be summarized as follows: (1) LGBM-CBFS performs better than LGBM_FS in the first few iterations when taking AUC or binary error as performance metrics, but it gradually approaches LGBM_FS during the subsequent training process. (2) LGBM-CBFS has a significant effect on model output at the beginning of the training process, but the effect has decreased as the number of iterations increases. (3) LGBM_FS leads the model to fail to converge (in the case of kdda) when sampling on high-dimensional sparse data with few key features due to quite a small probability assigned. In summary, we can learn from these experiments that, LGBM-CBFS outperforms the built-in strategy in LightGBM (uniform sampling), and it continuously improves the model quality during the training process.

4.2. Efficiency Evaluation. In this subsection, we report the experimental results with respect to efficiency upon all datasets. The average time cost (1e-3 seconds) for training one iteration on sparse datasets is shown in Figure 4. We set

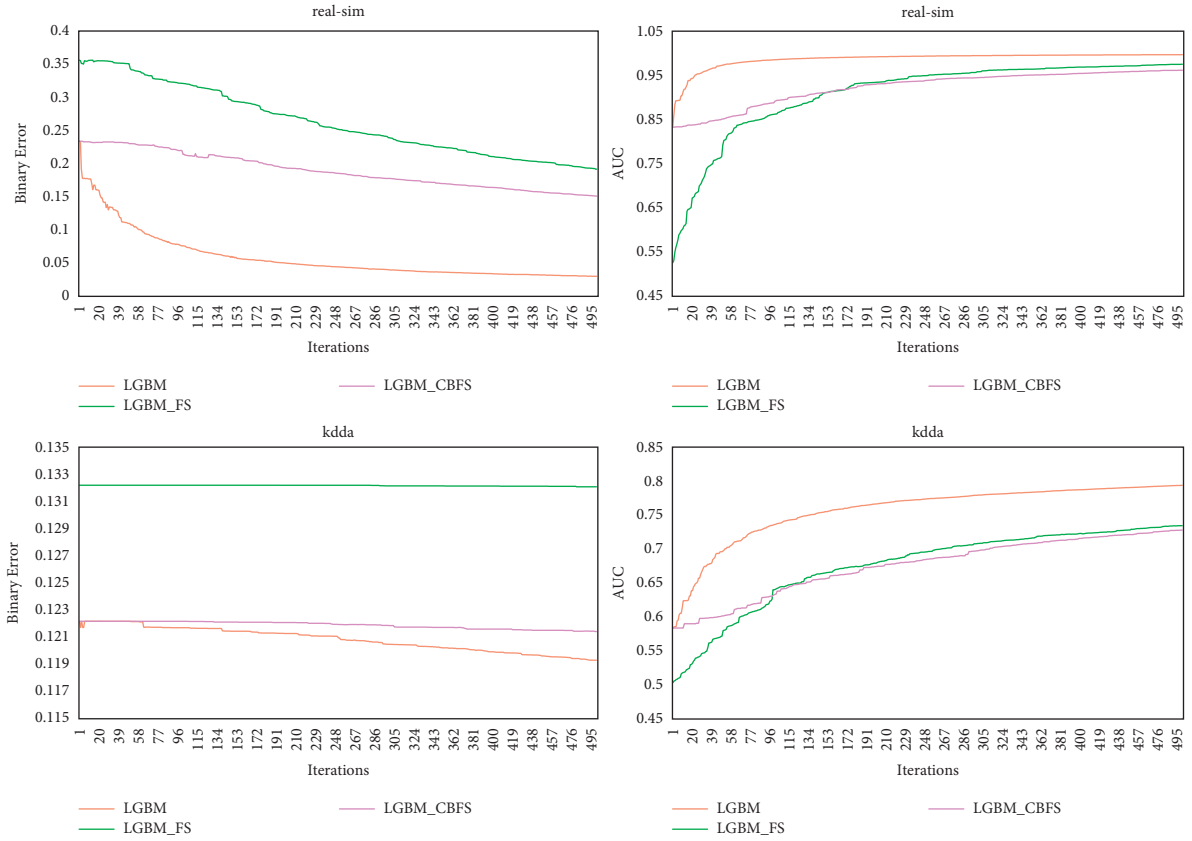

Figure 3: Iteration-accuracy curve.

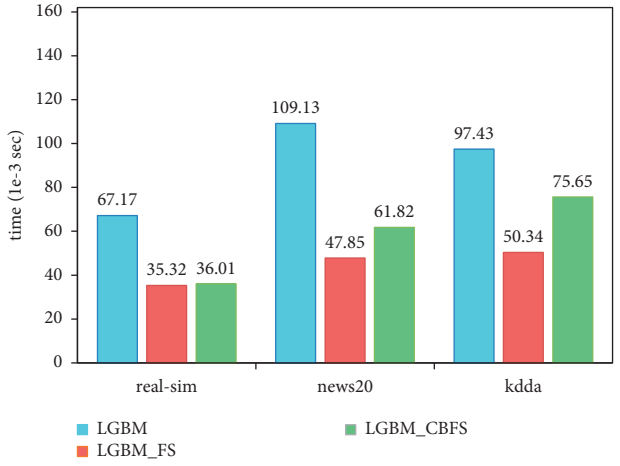

Figure 4: The average time cost for one iteration (sparse data).

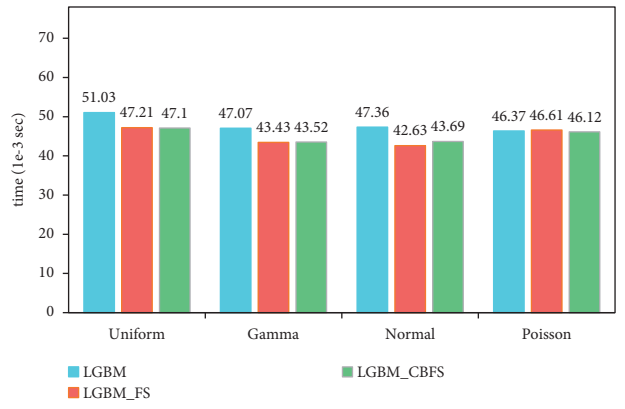

Figure 5: The average time cost for one iteration (dense data).

0.05% as the value of the sampling ratio. From the results, both LGBM_FS and LGBM-CBFS have accelerated the training process. LGBM-CBFS takes additional memory and computation cost because of the extra sort operation on features. It leads to more time overheads of training than LGBM_FS.

We use four artificial datasets descript in section 3.1 in our experiments to evaluate the training speed on dense datasets. We set 10% as the value of the sampling ratio. The experimental results in Figure 5 show that LGBM-CBFS and LGBM_FS slightly improve the computational efficiency in most situations. And there is no significant difference between LGBM-CBFS and LGBM_FS in time overheads.

In general, all these evaluations from accuracy and speed perspectives demonstrate that LGBM-CBFS could cut down the risk of training failure triggered by using a small sampling ratio; meanwhile, it often leads to higher accuracy in a high-dimensional sparse scenario.

5. Conclusions

In a real application context, training a GBDT model over a large-scale dataset is time-consuming. A natural idea to solve this issue is that sampling on input dataset to reduce data size, but it may lead model fail to converge in high-dimensional scenarios when using uniform sampling in LightGBM. In order to address this problem, we devised a heuristic feature sampling algorithm named LGBM-CBFS and implement it into LightGBM. Benefit from the strategy that chooses a representative subset of features based on the native importance scores of GDBT, LGBM-CBFS can heavily reduce excessive computation costs. We conduct a series of experiments and compare the proposed LGBM-CBFS algorithm with the built-in feature sampling algorithm in LightGBM. The experimental results show that LGBM-CBFS significantly improved the training effectiveness of GBDT over sparse data with high-dimensional features, meanwhile, without introducing additional computation overheads. In the future, we will explore potential optimization solutions of our sampling strategy for dense dataset scenarios.

Data Availability

The data are available on request by contacting the corresponding author (cse.HuiLi@gzu.edu.cn).

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was partially supported by the National Natural Science Foundation of China (62162010, 62162011, and 72161005) and Research Projects of the Science and Technology Plan of Guizhou Province (no. [2021]449).

References

[1] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[2] T. Chen and C. Guestrin, "XGBoost: a scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, New York, NY, United States, 2016.

[3] G. Ke, Q. Meng, T. Finley et al., "A highly efficient gradient boosting decision tree," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 3146–3154, Red Hook, NY, United States, 2017.

[4] H. Chernoff, "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations," *The Annals of Mathematical Statistics*, vol. 23, no. 4, pp. 493–507, 1952.

[5] W. Hoeding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 13–30, 1963.

[6] O. Watanabe, "Sequential sampling techniques for algorithmic learning theory," *Theoretical Computer Science*, vol. 348, no. 1, pp. 3–14, 2005.

[7] C. Domingo, R. Gavaldà, and O. Watanabe, *Adaptive Sampling Methods for Scaling up Knowledge Discovery Algorithms*, pp. 131–152, Discovery Science, 1999.

[8] C. Domingo, R. Gavaldà, and O. Watanabe, "Adaptive sampling methods for scaling up knowledge discovery algorithms," *Data Mining and Knowledge Discovery*, vol. 6, no. 2, pp. 131–152, 2002.

[9] O. Watanabe, *Simple Sampling Techniques for Discovery Science*, Technical Report, 2000.

[10] J. Chen and X. Chen, *A New Method for Adaptive Sequential Sampling for Learning and Parameter Estimation*, , pp. 220–229, Foundations of Intelligent Systems-international Symposium Springer-Verlag, 2011.

[11] J. Chen, "Scalable ensemble learning by adaptive sampling," in *Proceedings of the International Conference on Machine Learning and Applications*, vol. 1, pp. 622–625, Edinburgh, Scotland, 2012.

[12] P. Drineas, M. W. Mahoney, and S. Muthukrishnan, "Relative-error CUR matrix decompositions," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 2, pp. 844–881, 2007.

[13] A. Kyrillidis and A. Zouzias, "Non-uniform feature sampling for decision tree ensembles," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 4548–4552, Florence, Italy, 4-9 May 2014.

[14] J. H. Friedman, "Stochastic gradient boosting," *Computational Statistics & Data Analysis*, vol. 38, no. 2, pp. 367–378, 2002.

[15] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 13–30, 1963.

[16] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[17] R. E. Schapire, "The boosting approach to machine learning: an overview," *Nonlinear Estimation and Classification*, vol. 171, pp. 149–171, 2003.

[18] Z. Kalal, J. Matas, and K. Mikolajczyk, "Weighted sampling for large-scale boosting," in *Proceedings of the British Machine Vision Conference*, pp. 1–10, Leeds, UK, 2008.

[19] A. Ahmed El and M. W. Mahoney, *Fast Randomized Kernel Methods with Statistical Guarantees*, ArXiv http://abs/1411.0306, 2014.

[20] P. Drineas, M. M. Ismail, M. W. Mahoney, and D. P. Woodruff, "Fast approximation of matrix coherence and statistical leverage," *Journal of Machine Learning Research*, vol. 13, pp. 3475–3506, 2011.

[21] A. Rudi, D. Calandriello, L. Carratino, and L. Rosasco, "On fast leverage score sampling and optimal learning," in *Proceedings of the NeurIPS*, pp. 5677–5687, Montréal, Canada, 2018.

[22] D. Teng and S. Dasgupta, "Lifelong learning via online leverage score sampling," in *Proceedings of the Adaptive & Multitask Learning Workshop*, Long Beach, CA, USA, 2019.