软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(3):784–798 [doi: 10.13328/j.cnki.jos.005690]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

面向高维特征和多分类的分布式梯度提升树*

江佳伟[1], 符芳诚[1], 邵蓥侠[2], 崔 斌[1]

[1](高可信软件技术教育部重点实验室(北京大学),北京 100871)

[2](北京邮电大学 计算机学院,北京 100876)

通讯作者: 崔斌, E-mail: bin.cui@pku.edu.cn

摘 要: 梯度提升树算法由于其高准确率和可解释性,被广泛地应用于分类、回归、排序等各类问题.随着数据规模的爆炸式增长,分布式梯度提升树算法成为研究热点.虽然目前已有一系列分布式梯度提升树算法的实现,但是它们在高维特征和多分类任务上性能较差,原因是它们采用的数据并行策略需要传输梯度直方图,而高维特征和多分类情况下梯度直方图的传输成为性能瓶颈.针对这个问题,研究更加适合高维特征和多分类的梯度提升树的并行策略,具有重要的意义和价值.首先比较了数据并行与特征并行策略,从理论上证明特征并行更加适合高维和多分类场景.根据理论分析的结果,提出了一种特征并行的分布式梯度提升树算法 FP-GBDT.FP-GBDT 设计了一种高效的分布式数据集转置算法,将原本按行切分的数据集转换为按列切分的数据表征;在建立梯度直方图时,FP-GBDT 使用一种稀疏感知的方法来加快梯度直方图的建立;在分裂树节点时,FP-GBDT设计了一种比特图压缩的方法来传输数据样本的位置信息,从而减少通信开销.通过详尽的实验,对比了不同并行策略下分布式梯度提升树算法的性能,首先验证了 FP-GBDT提出的多种优化方法的有效性;然后比较了 FP-GBDT 与 XGBoost 的性能,在多个数据集上验证了 FP-GBDT 在高维特征和多分类场景下的有效性,取得了最高 6 倍的性能提升.

关键词: 梯度提升树;数据并行;特征并行;系统实现;算法比较

中图法分类号: TP311

中文引用格式: 江佳伟,符芳诚,邵蓥侠,崔斌.面向高维特征和多分类的分布式梯度提升树.软件学报,2019,30(3):784–798. http://www.jos.org.cn/1000-9825/5690.htm

英文引用格式: Jiang JW, Fu FC, Shao YX, Cui B. Distributed gradient boosting decision tree algorithm for high-dimensional and multi-classification problems. Ruan Jian Xue Bao/Journal of Software, 2019,30(3):784–798 (in Chinese). http://www.jos.org.cn/1000-9825/5690.htm

Distributed Gradient Boosting Decision Tree Algorithm for High-dimensional and Multi-classification Problems

JIANG Jia-Wei[1], FU Fang-Cheng[1], SHAO Ying-Xia[2], CUI Bin[1]

[1](Key Laboratory of High Confidence Software Technologies of Ministry of Education (Peking University), Beijing 100871, China)

[2](School of Computer Science, Beijing University of Posts and Telecommunications, Beijing 100876, China)

Abstract: Gradient boosting decision tree algorithm is widely used in various tasks, such as classification, regression, and ranking, owing to its high accuracy and strong interpretability. With the explosive growth of data volume, distributed gradient boosting decision tree algorithms have become an important research issue. Although there exists a series of implementations of distributed gradient boosting decision tree, they perform poorly on high-dimensional and multi-classification tasks. The data parallel strategy they adopt

* 基金项目: 国家自然科学基金(61832001, 61702015, 61702016); 国家重点研发计划(2018YFB1004403)

Foundation item: National Natural Science Foundation of China (61832001, 61702015, 61702016); National Key Research and Development Program of China (2018YFB1004403)

本文由智能数据管理与分析技术专刊特约编辑樊文飞教授、王国仁教授、王朝坤副教授推荐.

收稿时间: 2018-07-19; 修改时间: 2018-09-20; 采用时间: 2018-11-01

requires the transmission of gradient histograms, and this communication overhead becomes the bottleneck in many high-dimensional and multi-classification task. This study aims at this problem and tries to find an efficient parallel strategy that is more suitable for the target. Data-parallel and feature-parallel strategies are first compared based on a cost model, and it is theoretically proved that feature-parallel is more suitable for high-dimensional and multi-classification tasks. Based on the analysis, this paper proposes a feature-parallel distributed gradient boosting decision tree algorithm, named FP-GBDT. FP-GBDT designs an efficient distributed dataset transposition method to partition the training dataset by column. During the construction of gradient histogram, FP-GBDT uses a sparsity-aware method to accelerate the histogram construction. When splitting tree nodes, FP-GBDT develops a bitmap compression method to transmit the placement of instances, thereby reduces the communication overhead. This study compares the performance of distributed gradient boosting decision tree algorithm under different parallel strategies through extensive experiments. First, the effectiveness of proposed optimization methods in FP-GBDT is verified. Then, the representative of data-parallel strategy of FP-GBDT and XGBoost are compared. On various datasets, it is proved that FP-GBDT is more efficient in high-dimensional and multi-classification tasks. FP-GBDT achieves up to 6 times performance improvement than data-parallel implementations.

Key words: gradient boosting decision tree; data parallel; feature parallel; system implementation; performance comparison

梯度提升树(gradient boosting decision tree,简称 GBDT)是一种使用 Boosting 策略的集成机器学习算法.集成机器学习算法通常使用Bagging,Boosting,AdBoost等策略[1,2],基本思想是训练多个弱分类器来生成更准确的结果.GBDT 使用的 Boosting 策略通过依次学习多个弱学习器不断地提升性能,梯度提升树使用树模型作为弱学习器.梯度提升树算法在分类、回归、排序等诸多问题上取得了优异的性能[3−5],在学术界和工业界中被广泛地使用[6],也是例如 Kaggle 等数据竞赛中最受欢迎的算法之一.在大数据时代,由于单机的计算能力和存储能力有限,无法有效地处理动辄达到 TB 甚至 PB 级别的数据,因而需要在分布式环境下设计和执行梯度提升树算法,分布式梯度提升树算法因而成为目前的研究热点.

梯度提升树算法的训练主要包括 3 个关键过程.

1) 建立梯度直方图.对于给定的损失函数,对输入的训练数据,计算它们的梯度,将所有的梯度数据组织成梯度直方图的数据结构;
2) 寻找最佳分裂点.根据建立的所有特征的梯度直方图,计算出树节点的最佳分裂点,包括特征和分裂特征值;
3) 分裂树节点.根据计算出的最佳分裂点来分裂树节点,建立子节点,根据分裂点将训练数据划分到子节点上,并回到步骤 1)开始下一轮迭代.

在分布式环境下设计机器学习算法,有数据并行和特征并行两种策略:数据并行将训练数据集按行进行切分,每个计算节点处理一部分数据子集;而特征并行将训练数据集按列进行切分,每个计算节点处理一部分特征子集.已有的分布式梯度提升树系统,例如 MLlib[7],XGBoost[8],LightGBM[9,10]等,通常使用数据并行策略在分布式环境下训练.目前,没有工作系统性地比较数据并行和特征并行策略的优劣,数据并行是否在所有情况下都能取得最佳的性能,是一个仍然未被讨论和解决的问题.

笔者发现,在多种情况下,采用数据并行的分布式梯度提升树算法会遇到性能瓶颈:一方面,在梯度提升树算法所处理的多种问题中,多分类问题由于较高的计算复杂度和空间复杂度,采用数据并行的梯度提升树算法目前无法在分布式环境下有效地处理;另一方面,随着各行业的飞速发展,数据的来源越来越多,因而数据集的特征维度越来越高,这样的高维特征给数据并行的梯度提升树算法带来了新的挑战.为了解决以上存在的问题,本文关注在高维特征和多分类问题下如何高效地训练分布式梯度提升树算法.

在分布式环境下训练梯度提升树算法时,数据并行和特征并行采用不同的方式:数据并行策略将训练数据集按行切分到计算节点,计算节点建立局部梯度直方图,然后通过网络汇总计算节点的局部梯度直方图;特征并行策略将训练数据集按列切分到计算节点,计算节点建立梯度直方图并计算出最佳分裂点,然后通过网络交换分裂结果.由于梯度直方图的大小与特征数量和类别数量成正比,对于高维和多分类问题,数据并行需要通过网络传输大量的梯度直方图,因而存在性能不佳的问题.相比之下,特征并行需要通过网络传输的数据只与数据量大小有关,因而更适合高维和多分类的场景.

为了使用梯度提升树算法高效地求解高维和多分类问题,本文提出了一种基于特征并行的梯度提升树算法FP-GBDT.通过从理论上比较数据并行和特征并行,本文证明特征并行更适合高维和多分类场景,然后设计了特征并行的分布式梯度提升树算法.本文的贡献可以总结如下:在提出代价模型的基础上,从理论上比较了数据并行和特征并行两种训练策略的开销,证明对于高维和多分类数据集,特征并行具有较小的网络开销、内存开销和可扩展性;提出了基于特征并行的分布式梯度提升树算法FP-GBDT.首先,设计了一种分布式矩阵转置方法转换数据表征;然后,在建立梯度直方图时使用了稀疏感知的方法;最后,在分裂树节点时提出了比特图压缩方法来减少数据传输;通过与现有分布式梯度提升树算法进行详尽的实验比较,证明了 FP-GBDT 算法的高效性.

本文第 1 节介绍梯度提升树算法的预备知识和相关工作,包括算法模型和训练方法.第 2 节从理论上比较数据并行和特征并行的开销.第 3 节介绍 FP-GBDT 算法的细节.第 4 节给出实验对比结果.第 5 节总结全文.

1 预备知识及相关工作

在本节中,主要介绍研究对象梯度提升树算法的模型和训练方法.其中,第 1.1 节介绍数据集表征,第 1.2 节简要介绍梯度提升树算法,第 1.3 节介绍梯度提升树的训练方法,第 1.4 节介绍分布式梯度提升树的研究进展.

1.1 输入数据集

梯度提升树算法的输入数据是一组训练数据样本 $\{(x_i,y_i)\}_{i=1}^{N}$,这里,N 是数据样本的总数量,$x_i\in\mathbb{R}^D$ 是一个数据样本的特征,$y_i\in\mathbb{R}$是数据样本的标签.当数据样本是稠密时,x_i 被存储为一个数组;当数据样本是稀疏的,只需要存储 x_i 中的非零元素,也就是特征索引和特征值组成的键值对.

1.2 梯度提升树算法概述

梯度提升树算法是基于树的集成机器学习方法[8],梯度提升树算法会依次训练多个回归树模型.在一棵树中,从根节点开始,每个树节点需要给出一个分裂规则,此分裂规则包括分裂特征和分裂特征值,根据分裂规则将数据样本切分到下一层的子树节点,最终,每个数据样本 x_i 被分配到一个叶子节点,叶子节点将其上的数据样本预测为 w.与决策树不同之处在于:梯度提升树的叶子节点给出的是连续值,而不是类别,在训练完一棵树之后,将 w 加到每个数据样本的预测值上,然后以新的预测值开始训练下一棵树.梯度提升树累加所有树的预测值作为最终的预测值[1]:

$$\hat{y}_i=\sum_{t=1}^{T}\eta f_t(x_i),$$

其中,T 是树的数量,$f_t(x_i)$是第 t 棵树的预测结果,η是一个叫学习速度的超参数.

图 1 展示了梯度提升树算法的例子,目的是预测一个购物网站上用户的消费能力.

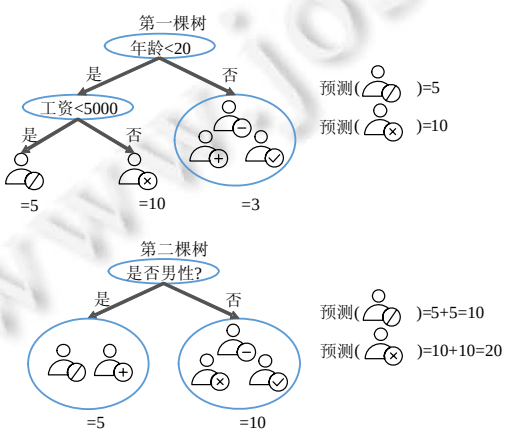

Fig.1 An example of gradient boosting decision tree

图 1 梯度提升树算法示例

第 1 棵树使用年龄是否小于 20 和工资是否小于 5 000 作为分裂规则,得到第 1 棵树的预测值,并以此更新数据样本的预测值;接着,以新的预测值开始训练第 2 棵树,第 2 棵树以是否为男性作为分裂规则,得到第 2 棵树的预测值.最终的预测值为两棵树的预测值之和,例如,标识为"/"的用户,预测值为 5 加上 5,等于 10.

1.3 训练方法

梯度提升树算法是有监督机器学习算法,其目标是最小化一个目标函数,对于梯度提升树来说,在建立第 t 棵树时,需要最小化以下的代价函数:

$$F^{(t)}=\sum_{i=1}^{N}l(y_i,\hat{y}_i^{(t)})+\Omega(f_t)=\sum_{i=1}^{N}l(y_i,\hat{y}_i^{(t-1)}+f_t(x_i))+\Omega(f_t).$$

这里,l 对于真实值和预测值给出损失,例如,logistic 损失 $l=\log(1+e^{-y_i\hat{y}_i})$,square 损失 $l=(y_i-\hat{y}_i)^2$.Ω是正则项,可以避免过拟合.依照 XGBoost 的选择[8],选择如下的正则项:

$$\Omega(f_t)=\gamma L+\frac{1}{2}\lambda\|w\|^2.$$

L 是树中叶子节点的数量,w 是叶子节点的预测值组成的向量,γ和λ是超参数.LogitBoost 算法用二阶近似来扩展 $F^{(t)}$:

$$F^{(t)}\approx\sum_{i=1}^{N}\left[l\left(y_i,\hat{y}_i^{(t-1)}++g_if_t(x_i)+\frac{1}{2}h_if_t^2(x_i)\right)\right]+\Omega(f_t).$$

这里,g_i 和 h_i 是数据的一阶和二阶梯度: $g_i=\nabla_{\hat{y}_i}l(y_i,\hat{y}_i),h_i=\nabla^2_{\hat{y}_i}l(y_i,\hat{y}_i)$.用 $I_j=\{i|x_i\in leaf_j\}$代表属于第 j 个叶子节点的数据样本的集合,去掉常数项之后,可以得到 $F^{(t)}$的近似表达:

$$\tilde{F}^{(t)}=\sum_{j=1}^{J}\left[\left(\sum_{i\in I_j}g_i\right)w_j+\frac{1}{2}\left(\sum_{i\in I_j}h_i+\lambda\right)w_j^2\right]+\gamma L.$$

根据以上的公式,第 j 个叶子节点的最佳预测值和最佳目标函数是:

$$w_j^*=-\frac{\sum_{i\in I_j}g_i}{\sum_{i\in I_j}h_i+\lambda},\tilde{F}^*=-\frac{1}{2}\sum_{j=1}^{J}\frac{(\sum_{i\in I_j}g_i)^2}{\sum_{i\in I_j}h_i+\lambda}+\gamma L.$$

以上的结果假设已经知道树的结构和数据样本的分布,可以遍历所有可能的树结构和数据样本分布来找到最佳的解.但是这种方法的计算复杂度太高,在实际中不可行.因此,现有的方法采用一种贪心的方法,从根节点开始逐层地分裂树节点,见算法 1.

算法 1. 训练梯度提升树的贪心分裂算法.

M:特征数量,N:数据样本数量,K:候选分裂点数量

g_i,h_i:一阶梯度和二阶梯度

for m=1 to M **do**

  产生 K 个候选分裂点 $S_m=\{s_{m1},s_{m2},\ldots,s_{mk}\}$

  扫描数据样本,建立有 K 个箱子的梯度直方图

  $G_{mk}=\sum g_i,H_{mk}=\sum h_i,s_{mk-1}<x_{im}<s_{mk}$

$g_{\max}=0,G=\sum_{i=1}^{N}g_i,H=\sum_{i=1}^{N}h_i$

for m=1 to M **do**

  G_L=0, H_L=0

  for k=1 to K **do**

    $G_L=G_L+G_{mk}$, $H_L=H_L+H_{mk}$

    $G_R=G-G_L$, $H_R=H-H_L$

$$g_{\max}=\max\left(g_{\max},\frac{G_L^2}{H_L+\lambda}+\frac{G_R^2}{H_R+\lambda}+\frac{G^2}{H+\lambda}\right)$$

输出增益最大的分裂结果

在计算一个树节点的最佳分裂点时,首先对每个特征计算出 K 个候选分裂点,然后扫描一遍所有数据样本,用梯度数据建立梯度直方图.例如:G_{mk} 对应第 m 个特征,累加所有对应特征值处在第 k–1 和第 k 个候选分裂点之间的数据样本的一阶梯度;H_{mk} 以同样的方式累加二阶梯度.建立完所有特征的梯度直方图之后,根据下面公式找到代价函数增益最大的分裂结果:

$$Gain=\frac{1}{2}\left[\frac{(\sum_{i\in I_L}g_i)^2}{\sum_{i\in I_L}h_i+\lambda}+\frac{(\sum_{i\in I_R}g_i)^2}{\sum_{i\in I_R}h_i+\lambda}-\frac{(\sum_{i\in I}g_i)^2}{\sum_{i\in I}h_i+\lambda}\right]-\gamma.$$

这里,I_L 和 I_R 代表分裂之后的左子节点和右子节点.选择候选分裂点有多种方式,精确的方法依据每个特征对所有数据样本进行排序,然后选择所有可能的分裂点.但这种方法需要反复地排序,在数据量很大时,计算开销和时间开销过于昂贵.另外一种叫分位数据草图的方法使用基于概率的近似数据结构,对每个特征产生对应特征值的近似分布[11].经典的分位数据草图的算法是 GK[12],基于 GK 的变种有 DataSketches[13]和 WQS[8]等.

梯度提升树算法中,两种避免过拟合的方法是衰减和特征下采样:衰减方法在累加树的预测值时乘以一个叫做学习速度的超参数 η;特征下采样方法对每棵树采样一个特征子集,这种方法被证明能够加快训练速度和提高模型的泛化能力.

1.4 分布式训练

目前已经有一些研究工作提出了分布式梯度提升树算法,例如 XGBoost[8],LightGBM[10],TencentBoost[14],DimBoost[18]等.XGBoost 由华盛顿大学的研究者提出,采用数据并行的训练模式,在单个计算节点上将数据集按列切分,以列存块的形式存储数据样本,为每个特征值增加一个指针指向其属于的数据样本,这样带来了很大的额外内存开销.XGBoost 提出了一系列的优化方法,包括一种对数据加权的分位数据草图算法、一种稀疏感知的寻找分裂点算法、一种将列存块存储到磁盘的块压缩和块切分的算法.LightGBM 由微软亚洲研究院的研究者提出,默认采用数据并行的训练模式,提出了深度优先的策略.实验证明,这能够提高准确率.除了数据并行之外,LightGBM 也实现了特征并行.但是这个简单的实现不支持数据集按列切分,需要在每个计算节点上存储完整的数据集,这在超大规模数据和真实的工业界环境中是不可行的.TencentBoost 和 DimBoost 由北京大学和腾讯联合设计,采用数据并行的训练模式,利用参数服务器[16−18]架构分布式存储算法模型,以支持高维度的数据集,在建立梯度直方图阶段提出了稀疏感知的算法,只需要读取数据样本中的非零特征;同时,用高效的并行机制加速梯度直方图的建立,在寻找最佳分裂点阶段提出了量化压缩算法减少梯度数据的传输.针对参数服务器的特殊架构提出了两阶段的方法,在参数服务器端计算局部最佳分裂点,在计算节点端汇总得到全局最佳分裂点.可以看到,目前的分布式梯度提升树算法的系统基本都采用数据并行的训练方式,或者实现的特征并行方式在实际中不可行,不能有效地处理本文针对的高维特征和多分类问题.

2 并行策略比较

将一个串行的机器学习算法并行化,首先需要设计如何将数据样本在多个计算节点之间分配.常用的并行策略包括数据并行和特征并行.假设训练数据集是一个矩阵,每一行是一个数据样本,数据并行按行切分数据集,这样,每个计算节点负责一个数据子集.与此相对应的,特征并行按列切分数据集,每个计算节点负责一个特征子集.

本节将针对内存开销和网络开销两个方面,从理论上比较运用两种并行策略的分布式梯度提升树算法,并针对本文研究的高维和多分类问题选择合适的并行策略.为了形式化地描述结果,本文中用到的符号定义见表1,其中,训练数据集是一个 $N\times D$ 的矩阵,N 是数据样本的数量,D 是数据样本的特征数量;集群中有 W 个计算节点,梯度提升树模型包含 T 棵树,每棵树有 L 层,每个特征的候选分裂点的个数是 q;对于多分类问题,用 C 来表示类

别数量.

Table 1 Definition of symbols

表 1 符号定义

符号	含义
N	数据样本的数量
D	数据样本的特征数量
W	集群中计算节点的数量
T	梯度提升树的树个数
L	每棵树的层数
q	每个特征候选分裂点的数量
C	数据样本类别

2.1 梯度直方图大小

梯度提升树算法的核心操作是建立梯度直方图和寻找分裂点.梯度直方图的大小与 3 个因素有关.

1) 特征维度 D.由于需要为每种特征建立两个梯度直方图(一阶梯度直方图和二阶梯度直方图),因而梯度直方图的总数量与 $2D$ 成比例;
2) 候选分裂点的数量 q.一个梯度直方图中箱子的数量等于候选分裂点的个数;
3) 类别数量 C.在多分类问题中,每个梯度数据是一个向量,表示在所有类别上的偏微分,因此,梯度直方图的大小与 C 成比例.二分类是一个特例,二分类时 C=1.

综上,一个树节点的梯度直方图大小为 S_h=8×2×D×q 字节,其中,8 字节表示一个双精度浮点数的大小.

2.2 内存开销

梯度提升树训练中的一个技巧是,只要树节点之间不存在数据重叠,就可以同时处理多个树节点,代价是存储更多的梯度直方图.对于目前普遍采用的逐层训练的方式,树的同一层的树节点之间不存在数据重叠,因而可以同时被处理,树的深度最大为 L,因此,同时处理的树节点的最大数量是 2^{L-1}.

使用数据并行策略时,每个计算节点需要建立所有待处理树节点上每个特征的直方图,因此,每个计算节点上梯度直方图的内存开销最多是 $S_h\times 2^{L-1}$.但是用特征并行时,每个计算节点只需要负责计算一部分特征的梯度直方图,因此内存开销是 $S_h\times 2^{L-1}/W$,与数据并行相比要低得多.

2.3 通信开销

当使用数据并行时,主要的通信开销是梯度直方图的汇总.尽管目前存在多种在分布式环境下聚合数据的分布式架构,例如 MapReduce[7],AllReduce[8],ReduceScatter[10],Parameter Server[14−17]等,但是每个计算节点至少需要通过网络传输本地存储的梯度直方图,因此,一棵树需要传输的梯度直方图的大小至少是 $S_h\times W\times(2^L-1)$.

当使用特征并行时,由于每个计算节点上存储着一个特征的全部特征值,因此,梯度提升树算法不需要在计算节点之间汇总梯度直方图.但由于按列切分数据集,在寻找最佳分裂点时,每个计算节点从本地存储的梯度直方图中计算出一个局部最佳分裂点,然后从所有计算节点中选择出一个全局最佳分裂点,这样的代价是只有一个计算节点知道数据样本的分裂结果,也就是数据样本被分到左子节点还是右子节点.这样,在完成树节点分裂之后,此计算节点需要向其他计算节点广播数据样本的位置(左子节点或者右子节点),带来了一定的通信开销.这部分的通信开销与数据样本的数量成正比,因此当树的深度增大时,通信量保持不变.综上,一棵树的总通信开销是 4×N×W×L 字节,其中,4 字节表示 32 位整型数的大小.这样简单地实现存在的问题是,直接用整型来发送数据样本的位置开销较大.为了解决这个问题,本文提出将数据样本的位置编码为比特图,每个比特代表一个数据样本的位置,0 代表左子节点,1 代表右子节点.这种方法能够显著降低通信开销,对于一个 L 层的树,总的通信开销是$\lceil N\times W\times L/8\rceil$字节.

2.4 分析结果总结

根据之前的分析结果,使用数据并行策略最多需要 $S_h\times 2^{L-1}$ 的内存开销和 $S_h\times W\times(2^L-1)$的通信开销,特征并

行需要 $S_h \times 2^{L-1}/W$ 的内存开销和$\lceil N \times W \times L/8 \rceil$的通信开销.

下面以公开数据集 RCV1-multi 为例,定量地比较两种策略的内存开销和通信开销.RCV1-multi 数据集有53.4 万个样本、4.7 万个特征和 53 个类别,使用 8 个计算节点建立深度为 7 的树,候选分裂点的个数设置为 100,每棵树上梯度直方图的大小是 3.7GB.当使用数据并行时,建立一棵树的内存开销最大是 118.4GB,通信开销最差情况下总共是 1.8TB;当使用特征并行时,建立一棵树的内存开销是 14.8GB,通信开销是 3.8MB.由此可见,对于高维和多分类问题,特征并行与数据并行相比,在通信开销和内存开销上明显更为高效.

3 算法介绍

本节首先从整体上介绍本文提出的算法 FP-GBDT,然后分别详细介绍 FP-GBDT 的关键技术.

3.1 整体介绍

图 2 是 FP-GBDT 整体工作流程,主要包括以下几个关键步骤.

(1) 训练数据集存储在分布式文件系统上,例如 HDFS,GFS,Ceph 等,默认的存储方式是按行切分.FP-GBDT 从分布式文件系统上加载训练数据集到计算节点,读入内存中;

(2) FP-GBDT 对训练数据集执行数据集转置操作,将原本按行切分的数据集转换为按列切分的方式,以特征行的方式存储在计算节点上;

(3) 每个计算节点独立地建立梯度直方图.为了有效地处理普遍存在的高维稀疏数据,本文提出了一种稀疏感知的方法来加速梯度直方图的建立;

(4) 建立完梯度直方图之后,每个计算节点从本地的梯度直方图中找出一个局部最佳分裂点,然后通过一个主节点从所有候选中找到全局最佳分裂点;

(5) 产生全局最佳分裂点的计算节点执行分裂树节点的操作,并产生数据样本在树中的位置;然后,使用一种比特图的压缩方法发送数据样本的位置给其他计算节点.

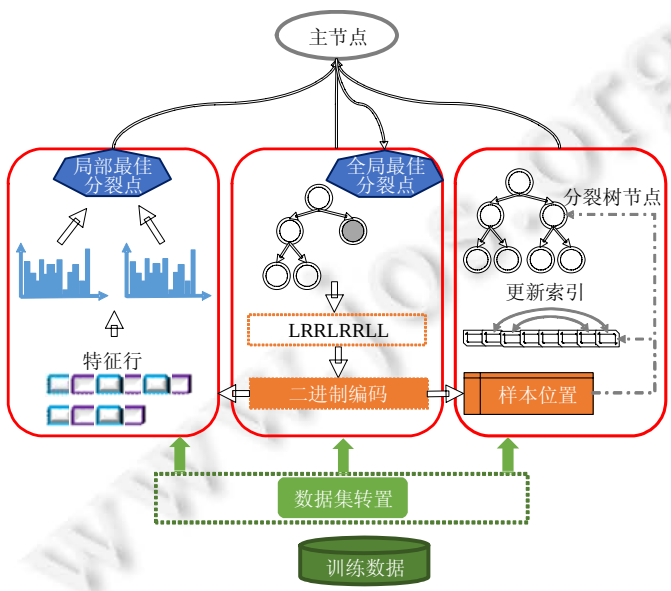

Fig.2 Framework of FP-GBDT

图 2 FP-GBDT 的架构

3.2 数据集转置

原始的训练数据集由多行组成,每行包含了一个数据样本的特征和标签.目前的数据集常常是稀疏格式,因

此经常将数据样本的特征存储为键值对的形式,其中,键是特征索引、值是特征值.数据集常常按行切分并存储在分布式文件系统上,例如 HDFS 等,这种存储格式天然不适合特征并行,因为特征并行需要在单个计算节点上汇总一个特征的所有特征值.为了解决这个问题,需要在梯度提升树算法执行之前对训练数据集进行转置操作,将按行存储的数据集转换为按列存储的方式.本文设计了一种高效的分布式数据集转置算法,如图 3 所示.

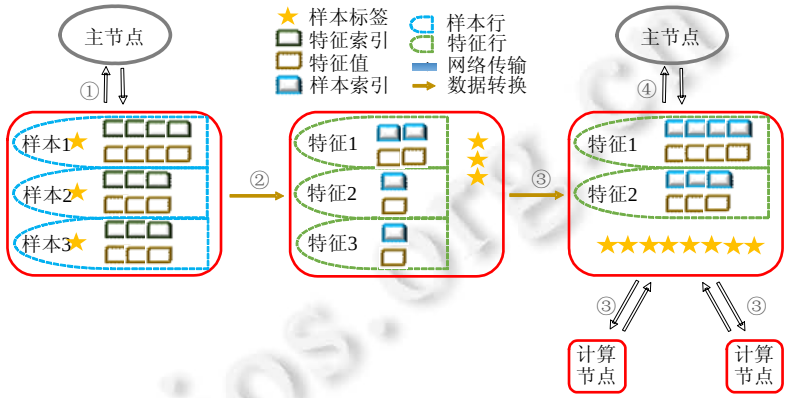

Fig.3 Distributed dataset transposition

图 3 分布式数据集转置

本方法的计算流程包括 4 个步骤.

(1) 计算数据样本的偏移.为了唯一标识每个数据样本,每个计算节点将本地数据样本的数量发送给主节点,然后,主节点从全局上统计数据样本的数量,给每个计算节点发送一个全局偏移值,限定了每个计算节点上数据样本标识范围.这样,每个数据样本的标识就等于计算节点的全局偏移加上本地偏移;

(2) 转置数据集.每个计算节点对本地的数据样本进行转置操作.原始的数据集由样本行组成,每个样本行存储着(特征索引,特征值)的键值对.本节的数据集转置方法将这种数据表示方式转换为特征行,每个特征行存储着一个特征出现的所有位置,也就是(样本索引,特征值)的键值对,表示此特征在每个数据样本中的非零值,这里的样本索引根据上一步产生的样本偏移计算而出;

(3) 重切分特征行.由于一个特征的特征值可能存在多个计算节点上,因而每个特征在多个计算节点上存在局部特征行.本方法通过 MapReduce 操作重切分所有计算节点上的局部特征行,使得每个特征的所有局部特征行在一个计算节点行进行汇总,形成此特征完整的特征行.通过这个操作,每个计算节点负责一个特征子集,存储着这个特征子集中所有特征的特征行,由(样本索引,特征值)的键值对组成;

(4) 广播样本标签.主节点从所有计算节点收集数据样本的标签,然后广播给计算节点.这些标签被每个计算节点用来计算数据样本的梯度.

3.3 建立梯度直方图

在数据集转置之后,每个计算节点使用特征行来建立梯度直方图.首先扫描特征行,为每个特征建立分位数据草图,然后使用数据草图产生候选分裂点,具体而言,使用一组 0~1 之间的分位值,例如{0,0.1,0.2,…,1.0},从分位数据草图中查询出每个特征的候选分裂值.

特征并行策略下,一个计算节点包含了某个特征所有出现的特征值,而数据并行策略下,一个计算节点只包含了某个特征的部分特征值.因此在特征并行策略下,一个树节点建立梯度直方图的操作比数据并行策略的计算量更大.由此可见:针对特征并行的特点,设计高效的机制来加速梯度直方图的建立是非常有意义的.

- 稀疏感知的建立梯度直方图方法.

建立梯度直方图需要直接处理数据样本,要达到优化的目标,应该考虑数据样本的特点.在实际应用中,很多真实的数据集常常是稀疏的,一个数据样本中只有一部分非常少的特征有非零值.传统的建立梯度直方图的

方法需要数据样本的所有特征值,包括值为零和值非零的特征,因此计算复杂度是 $O(N\times D)$,这里,N 是数据样本的数量,D 是特征数量.对于数据量大、特征维度高的情况,这样的计算复杂度过高.考虑到稀疏的数据特点和存在的问题,一个直观的想法是,能否利用数据稀疏性来加速梯度直方图的建立.基于这样的思路,本文提出一种稀疏感知的建立梯度直方图的技术,如图 4 所示,主要技术细节如下所述.

(1) 在读取数据样本之前,假设数据样本的所有特征值都为 0.一般情况下会认为特征值为非零是正常情况,特征值为 0 是异常情况,但真实情况是,数据样本中大部分特征值为 0,从概率的角度来看,特征值为 0 是大多数情况,而特征值为非零是极少数情况.基于这样的观察,本方法在读取数据样本之前,假设它们所有的特征值都为 0;

(2) 累加数据样本的梯度.经过数据集转置之后,每个计算节点上保存着所有数据样本的标签,使用数据样本的标签和预测值,计算数据样本的梯度,接着累加所有的梯度;

(3) 将梯度和增加到零桶中.一个特征的梯度直方图中,每个桶包含特征值的一个区间,这里将零桶定义为包含特征值为零的桶.本方法的第 1 步中假设所有特征值都为零,这样应该将每个数据样本的梯度放入零桶中,因此,本方法将第 2 步求得的梯度和加到每个特征的梯度直方图的零桶中;

(4) 重定位非零特征值.依次读取特征行存储的非零特征值,根据特征值的大小找到对应的梯度直方图的桶,然后将此特征值对应的数据样本的梯度加到此梯度直方图桶中;同时,为了保证梯度直方图的正确性,从零桶中减去此数据样本的梯度.

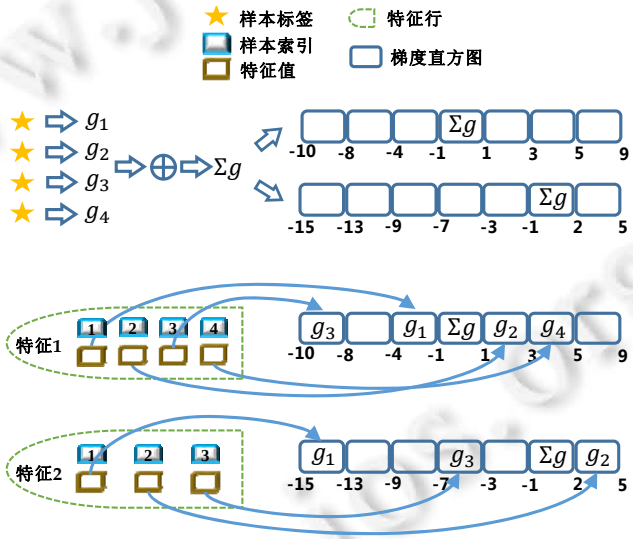

Fig.4 Sparsity-aware histogram construction

图 4 稀疏感知的建立梯度直方图方法

- 计算复杂度分析.

使用本文提出的稀疏感知建立梯度直方图的方法,首先需要扫描一遍数据样本计算梯度和,然后将梯度和加到每个特征的梯度直方图的零桶中,最后扫描一遍非零特征值.这样的计算复杂度是 $O\left(N+\frac{D}{W}+\frac{N\times d}{W}\right)$.这里,$N$ 是数据样本的数量,D 是特征的数量,W 是集群计算节点的数量,d 代表一个数据样本中非零特征值的平均数量.与传统方法的计算复杂度 $O(N\times D)$相比,由于 $d<<D$,因而计算复杂度降低.值得注意的是,在大规模数据场景下,一个问题是不同特征的稀疏程度不一致,不用特征上的非零数量可能差距很大,这会潜在地造成计算节点之间计算负载的不均衡.在 FP-GBDT 的数据集转置中,通过哈希的方式将整个特征集在计算节点之间分配,通过这种方式引入随机性,缓解计算节点的计算不均衡问题.在未来的工作中,将针对这个问题研究更加精细的

分配策略.

3.4 分裂树节点

找到一个树节点的最佳分裂点后,下一步是分裂树节点.使用特征并行策略时,某个计算节点产生了全局最佳分裂点,树节点的分裂结果(数据样本的位置)也只有此计算节点知道,因而,此计算节点需要将数据样本的位置广播给其他计算节点.为了减少通信开销,本文设计了一种比特图的方法来压缩数据样本的位置,如图 5 所示.

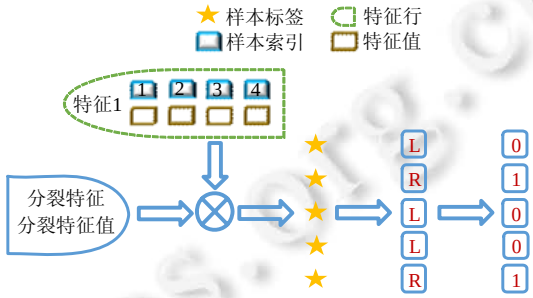

Fig.5 Bitmap compression

图 5 比特图压缩方法

- 比特图压缩.

如果使用整型的表达方式,广播数据样本位置的通信开销是 $4\times N\times W$ 字节.一个数据样本的位置是左子节点或者右子节点,因此,本方法使用一个比特图来二进制编码样本位置,左子节点编码为 0,右子节点编码为 1.使用这种方法,通信开销降低为 $N\times W/8$ 字节,带来了 32 倍的提升.当其他计算节点接收到比特图后,逐个比特地读取,然后更新数据样本在树中的位置.

除了能够节省通信开销,特征并行相比数据并行还有另外一个优势:当树的层数加深时,同层树节点的数量指数增长,数据并行策略下通信开销也会指数级增长;而使用特征并行策略时,树的一层的通信开销与树节点的数量无关,树深度的增加不会造成通信开销的增加.

4 实验对比

本节通过详尽的实验来验证 FP-GBDT 算法的有效性.

4.1 实验设置

- 原型实现.

FP-GBDT 在 Spark[19]的基础上实现,数据存储在 HDFS 上,机器学习任务提交到一个 8 节点的 Yarn[20]集群上.每个机器配置了 4 个 E3-1220 v3 CPU,32GB 内存和 1Gb 网络.

- 数据集.

本文的实验使用了 3 个数据集,见表 2.RCV1[21]和 RCV1-multi[22]是两个公开的文本分类数据集[23],样本数量分别为 69.7 万和 53.4 万,特征数量均为 4.7 万,RCV1 有 2 种类别,RCV1-multi 有 53 种类别.Synthesis 是一个人造数据集,包含 1 千万个样本和 10 万个特征.Synthesis 是一个二分类数据集,用来评估本文提出的优化方法的效果和特征维度的影响,RCV1 和 RCV1-multi 的特征相同而类别数量不同,被用来评估类别数量的影响.

- 基准方法.

如前所述,目前有多种分布式梯度提升树算法的实现,例如 Mllib,XGBoost 和 LightGBM 等.本文基于两个原因选择 XGBoost 作为数据并行策略的代表:第一,XGBoost 是目前使用最广泛的分布式梯度提升树系统;第二,XGBoost 有基于 Spark 实现的版本,因而实验对比是公平的.

- 超参数.

除非另外声明,对 RCV1 和 RCV1-multi 数据集选择 L=7,对 Synthesis 数据集选择 L=8,计算节点数量 W=10.对于其他超参数,候选分裂点数量 q=100,学习速度 η=0.1,特征采样率设为 1.0,树的数量设为 100.

Table 2 Datasets

表 2 数据集

数据集	大小(GB)	样本数量(万)	特征数量(万)	类别数量
RCV1	1.2	69.7	4.7	2
RCV1-multi	0.8	53.4	4.7	53
Synthesis	12	1 000	10	2

4.2 优化方法有效性

本节的实验用来评估本文提出的优化方法,验证它们的有效性,这些优化方法包括数据集转置、稀疏感知建立梯度直方图和比特图压缩.

- 数据集转置.

首先展示分布式数据集转置方法的性能,实验数据如图 6 所示,从 HDFS 上加载 3 个数据集的时间分别是 17s、12s 和 106s.使用简单的直接数据集转置,在 RCV1,RCV1-multi 和 Synthesis 数据集上需要的时间分别是 20s、13s 和 168s;而 FP-GBDT 执行数据集转置的操作,在 3 个数据集上的耗时分别是 9s、6s 和 95s,相比原始的数据集转置方法,速度得到了最高 2 倍的提升.虽然相比于数据并行的策略,FP-GBDT 带来了额外的时间开销,但是在后面的实验中将显示,FP-GBDT 显著提升了整体的性能.

- 稀疏感知建立梯度直方图.

当使用稀疏感知的方法时,对 Synthesis 数据集,建立树的根节点的梯度直方图的时间是 9s.但是当不使用这种方法时,在 1h 之内无法完成建立根节点的梯度直方图,原因是需要读取一个数据样本的所有特征.

- 比特图压缩.

找到最佳分裂点后,处理分裂树节点的任务时,FP-GBDT 广播数据样本的位置时使用比特图的压缩方法.本实验在 Synthesis 数据集上训练梯度提升树算法,建立一棵树的总时间开销中分裂树节点需要 18s;当使用了比特图压缩方法后,此部分时间开销降低为 8s,带来了 2 倍的速度提升.

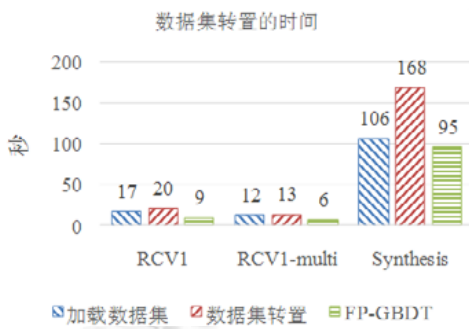

Fig.6 Effects of dataset transposition

图 6 数据集转置的效果

4.3 性能对比

4.3.1 FP-GBDT 与 XGBoost 的性能对比

本节的实验比较 FP-GBDT 与 XGBoost,FP-GBDT 使用特征并行策略,XGBoost 使用数据并行策略,下面分别评估特征数量、类别数量等因素对结果的影响.

- 特征数量的影响.

本文提出的 FP-GBDT 适合高维特征的数据集,为了评估特征数量对于实验结果的影响,本文使用 Synthesis

数据集的一部分特征子集,例如 Synthesis-25K 代表使用 Synthesis 数据集前 2.5 万个特征,Synthesis-100K 代表 Synthesis 数据集的全部特征.分别用 FP-GBDT 与 XGBoost 在 Synthesis 的多个数据子集上训练梯度提升树算法,FP-GBDT 与 XGBoost 对比的实验结果如图 7 所示.当特征数量从 2.5 万增加到 5 万和 10 万时,XGBoost 建立一棵树的时间从 80s 增加到了 144s 和 301s,性能分别下降了 1.8 倍和 3.7 倍.原因是特征数量增大 1 倍时,梯度直方图的大小也增加 1 倍,XGBoost 使用数据并行,通信开销线性增大,从而造成性能几乎线性下降;FP-GBDT 的性能下降较小,分别慢了 1.4 倍和 2.2 倍,FP-GBDT 使用特征并行,通信开销与特征数量无关,树的每一层的开销一样,因而只会受到计算开销增大的影响.总的来说,特征数量增大时 FP-GBDT 比 XGBoost 更加高效.

- 类别数量的影响.

如前文所分析,对于多分类问题,类别数量对梯度直方图的大小有直接的影响.为了更好地显示结果,本实验选择了两分类的 RCV1 数据集和 53 分类的 RCV1-multi 数据集,同时又将 RCV1-multi 的 53 类合并为 5 类.用 8 个计算节点对这 3 个数据集训练梯度提升树模型,图 8 显示了实验结果,统计了建立一棵树的平均时间.在 2 分类上的 RCV1 数据集上,XGBoost 平均需要 53s 训练一棵树,而 FP-GBDT 只需要 26s,训练速度快了 2 倍,所以 FP-GBDT 的收敛速度比 XGBoost 快得多.在 5 分类的 RCV1-multi 数据集上,XGBoost 需要 251s 训练一棵树,FP-GBDT 只需要 41s,快了 6.1 倍.类别增加到 5 类时,梯度直方图的大小增大了 5 倍,XGBoost 的性能下降了接近 5 倍,而 FP-GBDT 只慢了 1.5 倍,这说明 FP-GBDT 更适合多分类问题.当使用 53 类的 RCV1-multi 数据集时,FP-GBDT 训练一棵树需要 161s,而 XGBoost 出现了内存溢出(out of memory,简称 OOM)的错误,这证明了 XGBoost 使用的数据并行策略的内存开销较大.总的来说,FP-GBDT 比 XGBoost 更适合多分类任务,特别是类别数量较大的情况.

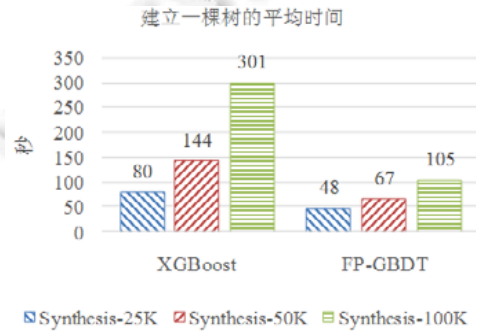

Fig.7 Impact of feature dimensionality

图 7 特征数量的影响

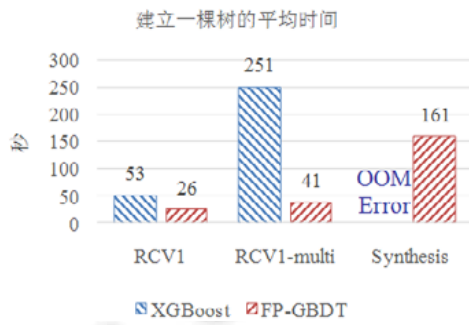

Fig.8 Impact of class number

图 8 类别数量的影响

- 树深度的影响.

接下来的实验将评估树的深度对性能的影响.一般来说,树的深度越大,训练时间越长,模型准确率更高.本实验在 RCV1 数据集上训练梯度提升树算法,逐步地增加树的深度,实验结果显示在图 9 和图 10 中.

深度增大时,XGBoost 和 FP-GBDT 的误差均下降,证明更深的树可以提高模型的准确率,代价是训练时间的增大.当树的深度从 8 增大到 9 时,数据并行策略下梯度直方图的大小增大 1 倍,因此,XGBoost 的运行速度慢了 2 倍;继续将树的深度增大到 10 时,XGBoost 出现了内存溢出的错误.

与此相对应,FP-GBDT 使用特征并行,能够高效地处理更深的树,树的深度增大时,每一层的通信开销保持不变,因此性能下降较为平稳,在可接受范围之内;同时,FP-GBDT 的内存开销较小,树的深度增加时没有出现资源不足的情况.

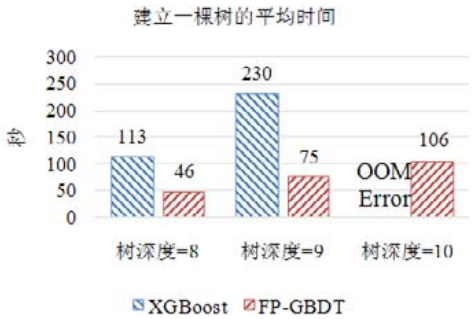

Fig.9 Impact of tree depth (time to build one tree)

图 9 树深度的影响(建立一棵树的平均时间)

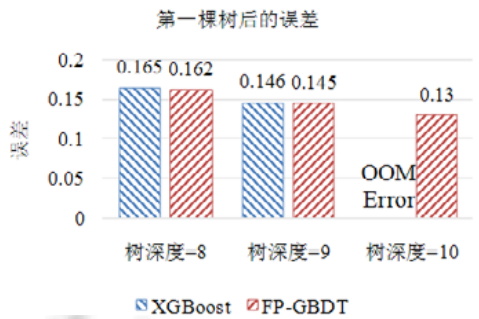

Fig.10 Impact of tree depth (error after the first tree)

图 10 树深度的影响(第 1 棵树后的误差)

4.3.2 FP-GBDT 与 LightGBM 的性能对比

如第 1.4 节所述,LightGBM 也实现了一个特征并行的梯度提升树算法,但是 LightGBM 的实现需要每个计算节点上已经存有完整的数据集,训练时,LightGBM 将整个数据集加载进内存,这样,LightGBM 不需要进行数据集转置的处理,也不需要在节点之间广播分裂后数据样本的位置.严格意义上,这不是一种真正的特征并行方式,在真实的应用中,单机的内存常常无法存储完整的数据集;在真实的环境中,数据集常常以存储在分布式文件系统上,LightGBM 在这种情况下无法工作.因此,LightGBM 的特征并行实现无法用在实际中.

但是为了给出更加全面的结果,本节使用 Synthesis 数据集比较了 FP-GBDT 和 LightGBM 的性能,实验设置与之前的实验保持一致.由于 LightGBM 无法从分布式文件系统读取数据,首先使用 hadoop fs 命令从 HDFS 上下载到每一个计算节点上,在所有计算节点下载完成之后,在每个计算节点上启动 LightGBM,从 HDFS 下载数据集加上本地读取数据的总时间是 LightGBM 的数据预处理的时间.

图 11 给出了实验结果,FP-GBDT 的数据预处理(包括加载数据集和数据集转置)时间是 201s,而 LightGBM 需要 820s 来对数据进行预处理,比 FP-GBDT 慢了 4 倍多;在训练阶段,FP-GBDT 建立一棵树的平均时间是 105s,LightGBM 需要 90s;在内存开销方面,LightGBM 由于需要在每个节点上存储完整的数据集,因此每台机器上消耗了 30GB 的内存,是 FP-GBDT 的内存开销的 6 倍.总体来看,LightGBM 单棵树的速度略快.这主要是因为每个计算节点加载完整的数据集,从而避免了通过网络传输数据样本的分裂位置;很多工业界的数据集大小常常达到 TB 级别,单个计算节点的内存显然无法存储这样大的数据集;对于存储在分布式文件系统上的数据集,LightGBM 首先需要将数据集下载到每个计算节点上,这带来了很大的开销.因此,对于较小的数据集和实验室级别的环境,并且资源较充裕时,LightGBM 是一个不错的选择;但是对于较大的数据集和真实的生产环境,LightGBM 常常是不可用的,而 FP-GBDT 的适用性很广,更加适合大数据时代的需求和发展趋势.

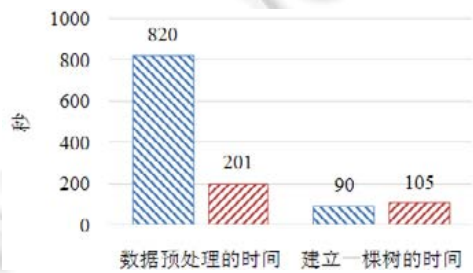

Fig.11 Performance comparison of FP-GBDT and LightGBM

图 11 FP-GBDT 与 LightGBM 的性能对比

5 总 结

本文研究面向高维特征和多分类问题的分布式梯度提升树算法.根据一个严格的代价模型,比较了数据并行策略和特征并行策略,证明了特征并行策略更适合高维和多分类场景;基于理论分析的结果,本文提出一种使用特征并行的分布式梯度提升树算法 FP-GBDT.FP-GBDT 首先利用对数据集进行分布式转置,转换为特征并行需要的数据表征;在建立梯度直方图时,FP-GBDT 设计了稀疏感知的方法;在分裂树节点时,FP-GBDT 使用一种比特图压缩的方法传输数据样本的位置.实验结果显示,FP-GBDT 与使用数据并行的 XGBoost 相比,性能的提升最高达到 6 倍以上,证明了特征并行的 FP-GBDT 在高维和多分类问题上的高效性.

References:

[1] Friedman J, Hastie T, Tibshirani R, *et al*. Additive logistic regression: A statistical view of boosting. The Annals of Statistics, 2000, 28(2):337−407.

[2] Friedman JH. Greedy function approximation: A gradient boosting machine. The Annals of Statistics, 2001, 1189−1232.

[3] Li P. Robust logitboost and adaptive base class (abc) logitboost. arXiv:1203.3491. arXiv Preprint, 2012.

[4] Burges CJ. From ranknet to lambdarank to lambdamart: An overview. Learning, 2010,11(23-581):81.

[5] Stephen T, Weinberger KQ, Agrawal K, Paykin J. Parallel boosted regression trees for Web search ranking. In: Proc. of the 20th Int'l Conf. on World Wide Web. 2011. 387−396.

[6] He X, Pan J, Jin O, Xu T, Liu B, Xu Tao, Shi Y, Atallah A, Herbrich R, Bowers S, *et al*. Practical lessons from predicting clicks on ads at facebook. In: Proc. of the 8th Int'l Workshop on Data Mining for Online Advertising. 2014. 1−9.

[7] Meng X, Bradley J, Yavuz B, Sparks E, Venkataraman S, Liu D, Freeman J, Tsai DB, Amde M, Owen S, *et al*. Mllib: Machine learning in apache spark. The Journal of Machine Learning Research, 2016,17(1):1235−1241.

[8] Chen TQ, Guestrin C. Xgboost: A scalable tree boosting system. In: Proc. of the 22nd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2016. 785−794.

[9] Meng Q, Ke GL, Wang TF, When W, Ye QW, Ma ZM, Liu TY. A communication-efficient parallel algorithm for decision tree. In: Proc. of the Advances in Neural Information Processing Systems. 2016. 1279−1287.

[10] Ke GL, Meng Q, Finley T, Wang TF, Chen W, Ma WD, Ye QW, Liu TY. Lightgbm: A highly efficient gradient boosting decision tree. In: Proc. of the Advances in Neural Information Processing Systems. 2017. 3149−3157.

[11] Karnin Z, Lang K, Liberty E. Optimal quantile approximation in streams. In: Proc. of the 57th IEEE Annual Symp. on Foundations of Computer Science (FOCS). 2016. 71−78.

[12] Greenwald M, Khanna S. Space-efficient online computation of quantile summaries. ACM SIGMOD Record, 2001,30:58−66.

[13] Yahoo. Data sketches. https://datasketches.github.io/

[14] Jiang J, Jiang JW, Cui B, Zhang C. Tencentboost: A gradient boosting tree system with parameter server. In: Proc. of the 2017 IEEE 33rd Int'l Conf. on Data Engineering. 2017. 281−284.

[15] Jiang JW, Cui B, Zheng C, Fu FC. DimBoost: Boosting gradient boosting decision tree to higher dimensions. In: Proc. of the 2018 Int'l Conf. on Management of Data. 2018. 1363−1376.

[16] Abadi M, Barham P, Chen JM, Chen ZF, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, *et al*. Tensorflow: A system for large-scale machine learning. OSDI, 2016,16:265−283.

[17] Jiang JW, Cui B, Zhang C, Yu LL. Heterogeneity-aware distributed parameter servers. In: Proc. of the 2017 ACM Int'l Conf. on Management of Data. 2017. 463−478.

[18] Li M, Andersen DG, Park JW, Smola AJ, Ahmed A, Josifovski V, Long J, Shekita EJ, Su BY. Scaling distributed machine learning with the parameter server. OSDI, 2014,14:583−598.

[19] Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proc. of the 9th USENIX Conf. on Networked Systems Design and Implementation. 2012. 2.

[20] Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H, Seth S, *et al*. Apache hadoop yarn: Yet another resource negotiator. In: Proc. of the 4th Annual Symp. on Cloud Computing. 2013. 5.

[21] RCV1 dataset. https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#rcv1.binary

[22] RCV1-Multi dataset. https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html#rcv1.multiclass

[23] Lewis DD, Yang YM, Rose TG, Li Fan. Rcv1: A new benchmark collection for text categorization research. Journal of Machine Learning Research, 2004,5:361−397.

江佳伟(1990－),男,湖北洪湖人,博士,CCF 学生会员,主要研究领域为机器学习.

符芳诚(1996－),男,学士,主要研究领域为机器学习.

邵蓥侠(1988－),男,博士,副研究员,博士生导师,CCF 专业会员,主要研究领域为数据库,知识图谱数据管理,并行图计算,知识工程.

崔斌(1975－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为数据库,大数据管理分析.