XGBoost：一种可伸缩的树型增强系统

陈天奇
华盛顿大学
tqchen@cs.washington.edu

Carlos Guestrin
华盛顿大学
guestrin@cs.washington.edu

摘要

树推动是一种高效且广泛使用的机器学习方法。 在本文中，我们描述了一种名为XGBoost的可扩展端到端树型增强系统，该系统被数据科学家广泛使用，以实现许多机器学习挑战的最新成果。 我们提出了一种新颖的稀疏数据稀疏感知算法和加权分位数草图近似树学习。 更重要的是，我们提供有关缓存访问模式，数据压缩和分片的见解，以构建可扩展的树型增强系统。 通过结合这些见解，XGBoost可以使用比现有系统少得多的资源来扩展数十亿个示例。

关键词

大型机器学习

1. 介绍

机器学习和数据驱动方法在许多领域变得非常重要。智能垃圾邮件分类器通过学习大量垃圾邮件数据和用户反馈来保护我们的电子邮件； 广告系统学会将正确的广告与正确的背景相匹配； 欺诈检测系统保护银行免受恶意攻击者的侵害 异常事件检测系统帮助实验物理学家找到导致新物理学的事件。 推动这些成功应用的两个重要因素是：使用捕获复杂数据依赖性的有效（统计）模型以及从大型数据集中学习感兴趣模型的可扩展学习系统。

在实践中使用的机器学习方法中，渐变树增强[10][1]是许多应用中的一种技术。 已经显示树促进在许多标准分类基准上给出了最新的结果[16]。 LambdaMART [5]，这是树型的一种变种，可以实现排名的最新结果问题。 除了被用作独立预测器之外，它还被纳入真实世界的生产流水线中用于广告点击率预测[15]。 最后，这是集合方法的事实选择，并用于诸如Netflix奖等挑战[3].

在本文中，我们描述了XGBoost，一种可扩展的树木增强机器学习系统。 该系统可作为开源软件包提供[2]。系统的影响在许多机器学习和数据挖掘挑战中得到了广泛的认可。 以机器学习竞赛网站Kaggle举办的挑战为例。 在29个挑战获胜的解决方案中[3] 在2015年期间发表于Kag-s博客，17个解决方案使用XGBoost。 在这些解决方案中，八个单独使用XGBoost来训练模型，而其他大多数人则将XGBoost与神经网络合并在一起。 为了比较，第二种最流行的方法，即深度神经网络被用于11种解决方案。 该系统的成功也见于2015年KDDCup，其中XGBoost被每一个获胜团队使用，
10. 此外，胜利小组报告说，集成方法的性能仅超过配置良好的XGBoost [1].

这些结果表明，我们的系统在广泛的问题上提供了最先进的结果。 这些获奖解决方案中存在的问题包括：商店销售预测； 高能物理事件分类； 网页文本分类； 顾客行为预测； 运动检测； 广告点击率预测； 恶意软件分类； 产品分类； 危害风险预测； 大规模在线课程辍学率预测。 尽管依赖于领域的数据分析和特征工程在这些解决方案中发挥了重要作用，但XGBoost是学习者的一致选择，这一事实表明了我们的系统和树木推动的影响和重要性。

XGBoost成功背后的最重要因素是其在所有情况下的可扩展性。 该系统的运行速度比单台计算机上现有流行解决方案快十倍以上，并可扩展到分布式或内存有限设置的数十亿个示例。 XGBoost的可扩展性是由于几个重要的系统和算法优化。 这些创新包括：一种新颖的树学习算法用于处理稀疏数据； 理论上合理的加权分位素描程序能够在近似树学习中处理实例权重。 并行和分布式计算使得学习速度更快，从而可以更快地进行模型探索 更重要的是，XGBoost利用外核

[1] 梯度增强树也被称为梯度增强机（GBM）或梯度增强回归树（GBRT）

[2] https://github.com/dmlc/xgboost

[3] 解决方案来自每场比赛的前三名球队。

KDD'16，2016年8月13 - 17日，美国加利福尼亚州旧金山

DOI：http://dx.doi.org/10.1145/2939672.2939785

计算并使数据科学家能够在桌面上处理数百万个示例。最后，将这些技术结合起来构建一个端到端系统，以最少量的集群资源扩展到更大的数据更加令人兴奋。本文主要贡献如下：

- 我们设计和构建高度可扩展的端到端树木增强系统。
- 我们提出了一个理论上加权的分位数草图，用于有效的提案计算。
- 我们引入了一种新颖的并行树学习稀疏感知算法。
- 我们提出了一种有效的用于核外树学习的缓存感知块结构。

尽管在并行树增强方面有一些现有的工作[22, 23, 19]，诸如核外计算，缓存感知和稀疏感知学习等方向尚未被探索。 更重要的是，将所有这些方面结合起来的端到端系统为现实世界的用例提供了一种新颖的解决方案。这使得数据科学家和研究人员能够构建树推进算法的强大变体[7, 8]。 除了这些主要贡献之外，我们还在提出一个正规化的学习目标方面作出了进一步的改进，我们将包括这些目标的完整性。

在本文的其余部分安排如下。 我们将首先回顾一下树推广，并在第二部分介绍一个正规化的目标。2。 然后我们在第二部分描述分裂发现方法。3 以及Sec。中的系统设计。4包括相关的实验结果，为我们描述的每个优化提供量化支持。 相关工作在第二节讨论。5。 详细的端到端评估包含在第2节中。6。 最后，我们在第二节中总结这篇论文。7.

2. 树在一个坚果

我们在本节回顾梯度树增强算法。 推导出现于现有文献中梯度提升的相同思想。 特别地，二阶方法起源于Friedman等人。 [12]。 我们对规范化目标进行了小幅改进，这些目标在实践中发现很有帮助。

2.1 正规学习目标

对于具有n个示例和m个特征的给定数据集

$D = \{ (x_i, y_i) \} (|D| = n, x_i \in R, y_i \in R)$，树ensem-

ble模型（如图2所示）1）使用K的附加功能预测输出。

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in F, \qquad (1)$$

$$F \{ \} \to \in$$

哪里 $= f(x) = w_{(q(x))}$ $(q: R^{\ } \quad T, \ w \ R^{\ })$ 是回归树（也称为CART）的空间。 这里q rep-

不满意将每个树映射到相应叶索引的结构。 T是树中的叶子数量。 每个f对应一个独立的树结构q和叶子权重w。与决策树不同，每个回归树包含每个叶子上的连续分数，我们使用wi表示第i个叶子上的分数。 对于一个给定的例子，我们
将使用树中的决策规则（由q给出）进行分类

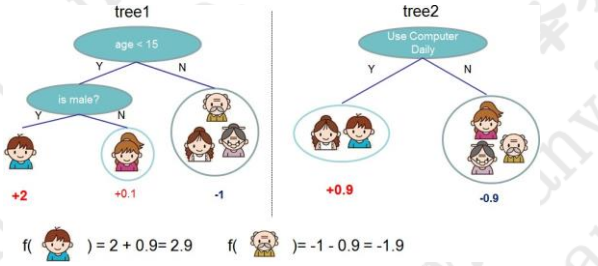


图1：树集成模型。 给定示例的最终预测是每棵树的预测总和。

它进入树叶并通过总结相应树叶中的得分（由w给出）来计算最终预测。 要学习模型中使用的一组函数，我们将以下正则化目标最小化。

$$L(\varphi) = \sum_i l(y_i, y_i) + \sum_k \Omega(F_k) \qquad (2)$$

$$\text{其中} \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

这里l是一个可微分的凸损失函数，用来衡量预测yi和目标yi之间的差异。 第二项Ω惩罚模型的复杂性（即回归树函数）。 额外的正则化术语有助于平滑最终学习权重以避免过度拟合。 直观地说，正则化目标往往会选择一个采用简单和预测功能的模型。 正规化技术已被用于正规化贪婪森林（RGF）[25]模型。 我们的目标和相应的学习算法比RGF更简单，并且更容易并行化。当正则化参数设置为零时，目标回落到传统梯度树增强。

2.2 渐变树提升

方程中的树集合模型。 （2）包括作为参数的功能，并且不能使用欧几里得空间中的传统优化方法来优化。 相反，模型是以加性方式进行训练的。 形式上，让 $\hat{y}_i^{(t)}$ 成为
我们将在第t次迭代中预测第i个实例
需要添加英尺以最小化以下目标。

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

这意味着我们贪婪地添加最能改进我们的f

根据公式 （2）。 二阶近似可用于在一般环境中快速优化目标[12].

$$\sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

其中 $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ 和 $h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ 是对损失函数的一阶和二阶梯度统计量，
灰。 我们可以删除常数项以在步骤t获得以下简化目标。

$$\sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \qquad (3)$$

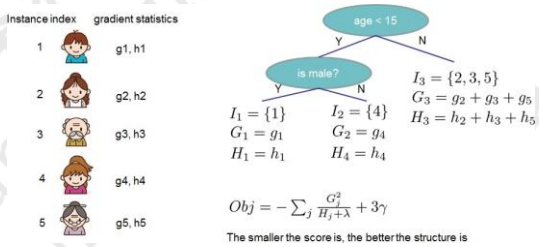


图2：结构分数计算。 我们只需要总结梯度和二阶gra-每个叶子上的统计数据，然后应用评分公式来获得质量得分。

定义Ij = {i|q(xi) = j}作为叶j的实例集。 我们可以重写Eq（3）通过扩展Ω如下

$$\tilde{L}^{(t)} = \sum_{j=1}^{T} [(\sum_{i\in I_j} g_i) w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda) w_j^2] + \gamma T \quad (4)$$

对于固定结构q（x），我们可以计算出最优值叶j的重量w∗j

$$w_j^* = -\frac{\sum_{i\in I_j} g_i}{\sum_{i\in I_j} h_i + \lambda}, \quad (5)$$

并通过计算相应的最优值

$$\tilde{L}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T} \frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T. \quad (6)$$

公式（6）可以用作评分函数来衡量树结构的质量q。 这个分数就像用于评估决策树的杂质分数一样，除了它是为更广泛的目标函数而推导的。图。2 说明了如何计算这个分数。

通常不可能枚举所有可能的树结构q。 改为使用从单个叶子开始并反复将分支添加到树中的贪婪算法。假设IL和IR是分割后的左右节点的实例集合。 假设I = IL ∪ IR，则分裂后的损耗减少量由下式给出

$$L_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \quad (7)$$

这个公式通常用于评估拆分候选人。

算法1：分割查找的精确贪婪算法

输入： I，当前节点的实例集
输入： d，特征尺寸
增益←0
G← $\sum_{i\in I} g_i$, *H←* $\sum_{i\in I} h_i$
对于k = 1到m来说
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  对于j中的排序（I，x_{jk}）来说
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    分数←max（分数， $\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}$）
  结束
结束
输出： 以最高分数分割

算法2：分割查找的近似算法

对于k = 1到m来
  根据特征k上的百分点提出$S_k = \{s_1, s_2, \ldots s_l\}$。
  建议可以按树（全局）或每个分割（本地）完成。
结束
对于k = 1到m来说
  $G_{kv} \leftarrow \sum_{j\in\{j \mid s_k, v \geq x_{jk} > s_k, v-1\}} g_j$
  $H_{kv} \leftarrow \sum_{j\in\{j \mid s_k, v \geq x_{jk} > s_k, v-1\}} h_j$
结束
按照上一节中的相同步骤查找最大值只在提议的分割中得分。

13]，它在一个商业软件TreeNet中实现[4]用于渐变增强，但在现有中未实施

开源软件包。 根据用户反馈，使用色谱柱子采样可以防止过度拟合
传统的行子采样（这也是支持的）。 列子样本的使用也加速了稍后描述的并行算法的计算。

3. 分割查找算法

3.1 基本的精确贪婪算法

树学习中的关键问题之一是找到最好的分裂，如方程（7）。 为此，分割查找算法列举了所有特征上所有可能的分割。 我们称之为精确的贪婪算法。
大多数现有的单机树增强实现 -

s，比如scikit-learn [20]，R's gbm [21]以及XGBoost的单机版本支持精确的贪婪算法，

2.3 收缩和柱子采样

除了第二节中提到的正常化目标之外。2.1，还使用了另外两种技术来进一步防止过度配合。 第一种技术是Friedman [11]。 收缩比例在每次增加树木的步骤后新增加一个因子 η 的权重。 类似于抽象优化的学习率，收缩会减少每棵树的影响，并为未来的树留下空间，

证明模型。 第二种技术是列（特征）
二次抽样。 这项技术被用于RandomForest [4,

rithm。 Alg中显示了精确的贪婪算法。1。 枚举连续特征的所有可能分割在计算上要求很高。 为了有效地这样做，该算法必须首先根据特征值对数据进行排序，并按照排序顺序访问数据以累积结构分数的梯度统计量，方程式（7).

3.2 近似算法

精确的贪婪算法非常强大，因为它贪婪地枚举了所有可能的分割点。 但是，当数据不完全适合内存时，不可能有效地做到这一点。 同样的问题也出现在dis-

\的https：//www.salford-systems.com/products/treenet

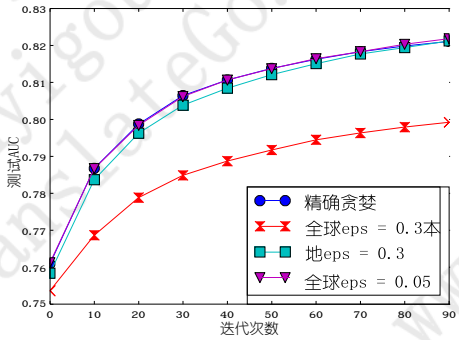


图3：希格斯10M数据集上测试AUC收敛的比较。eps参数对应于近似草图的准确度。 这个大致转换为提案中的1 / eps存储桶。 我们发现本地提案需要更少的桶，因为它提炼了分裂的候选人。

贡献的设置。 为了支持这两个设置中的有效梯度树推进，需要一个近似算法。

我们总结了一个近似的框架，类似于过去文献中提出的观点[17, 2, 22]，在阿尔格。2。 总之，该算法首先根据特征分布的百分位提出候选分裂点（具体标准将在第二节中给出）。3.3）。 该算法然后将连续特征映射到由这些候选点分割的桶中，聚合统计数据并基于汇总统计找到提议中的最佳解决方案。

该算法有两种变体，具体取决于何时提出提案。 全局变体提出了树木建设初始阶段的所有候选分割，并使用相同的提议在各级进行分裂查找–秒。 本地变体在每次拆分后重新提出。 全局方法比本地方法需要更少的提议步骤。 然而，全球提案通常需要更多的候选点，因为候选人在每次拆分后都不会被完善。 当地方案在分裂之后改进了候选人，并且可能更适合深层树木。 希格斯玻色子数据集上不同算法的比较如图1所示。3。 我们发现当地的提案确实需要更少的候选人。 全球提案可以与当地提供的候选人一样准确。

大多数现有的分布式树学习近似算法也遵循这个框架。值得注意的是，也可以直接构造梯度统计的近似直方图[22]。 也可以使用分箱策略的其他变体而不是分位数[17]。 分位策略从分配和重新计算中受益，我们将在下一小节详细介绍。 从图中看出，3，我们还发现分位数策略可以得到与给定合理近似水平的精确贪婪相同的精度。

我们的系统能够高效地支持单机设置的精确贪婪，以及针对所有设置的本地和全球提案方法的近似算法。用户可以根据自己的需要自由选择方法。

3.3 加权分位素描

近似算法中的一个重要步骤是提出候选分割点。通常使用特征的百分位数来使候选人平均分布在da-

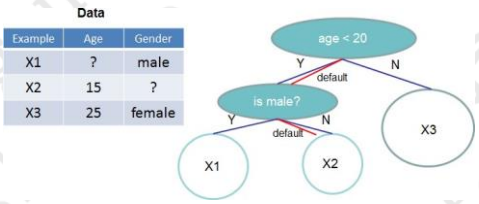


图4：具有默认方向的树结构 当缺少拆分所需的功能时，会将示例分类为默认方向。

TA。 形式上，设成多集$D_k = \{(x_{1k}, h_1), (x_{2k}, h_2) \ldots (x_{Nk}, h_N)\}$表示每个训练实例的第k个特征值和第2阶梯度统计量。我们可以定义一个秩函数$r_1: R \to [0, +\infty)$为

$$r_1(z) = \frac{1}{\sum_{(x,h)\in D_k} h} \sum_{(x,h)\in D_k, x<z} h, \quad (8)$$

它代表了其特征实例的比例值k比z小。 目标是找到候选分割点$\{s_1, s_1, \ldots s_{kl}\}$，这样

$$|r_1(s_{k,j}) - r_1(s_{k,j+1})| < \varepsilon, \quad s_1 = \min_i x_{ik}, \quad s_{kl} = \max_i x_{ik} \quad (9)$$

这里E是一个近似因子。 直观地说，这意味着大致有1 / E个候选点。 这里每个数据点由hi加权。 为了明白为什么hi代表重量，我们可以重写Eq（3）为

$$\sum_{i=1}^{n} \frac{h_i (f_t(x_i) - g_i/h_i)^2}{2} + \Omega(f_t) + \text{常数},$$

这是具有标签gi / hi和权重hi的精确加权平方损失。对于大型数据集，找到符合标准的候选分割并不重要。

当每个实例具有相同的权重时，称为分位数草图[14, 24]解决了这个问题。 但是，加权数据集不存在分位数草图。 因此，大多数现有的近似算法要么对有可能失败或启发式的数据的随机子集进行排序，而是没有理论保证。 为了解决这个问题，我们引入了一种新的分布式加权分位数草图算法，可以处理加权数据并具有可证明的理论保证。 总体思路是提出一个支持合并和修剪操作的数据结构，每个操作都被证明可以保持一定的精确度。 补充材料中给出了该算法的详细说明以及证明[5]（链接在脚注中）。

3.4 稀疏感知分割查找

在许多现实世界的问题中，输入x很稀少是很常见的。 稀疏性有多种可能的原因：1）数据中存在缺失值； 2）统计中频繁的零条目； 和3）诸如单热编码之类的特征工程的伪像。 使算法知道数据中的稀疏模式非常重要。 为此，我们建议在每个树节点中添加一个默认方向，如图3所示。4。 当稀疏矩阵x中缺少值时，实例为

[5] 链接到补充材料
http://homes.cs.washington.edu/~tqchen/pdf/xgboost-supp.pdf

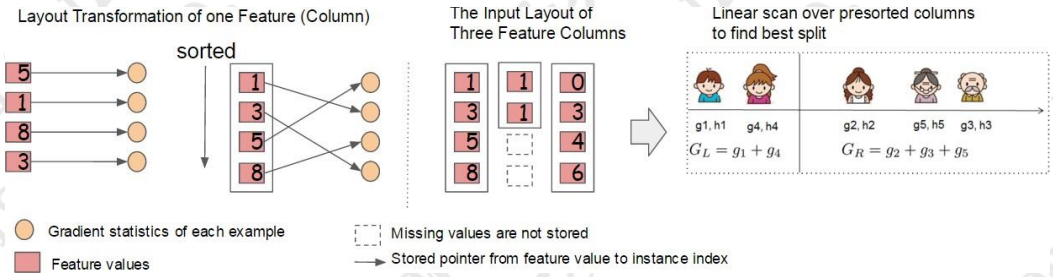


图6：并行学习的块结构 块中的每一列都按相应的特征值排序。 对块中的一列进行线性扫描足以枚举所有分割点。

算法3： 稀疏感知分割查找

输入： I，当前节点的实例集输入：$I_k = \{i \in I \mid x_{ik} \neq \text{缺失输入}\}$:
d，特征尺寸
也适用于近似设置，仅收集桶中未丢失条目的统计数据

增益←0
G← $\sum_{i \in I,\ gi,\ H}$ *我∈*

对于k = 1到m来说
//列举缺失值转到正确
GL←0，HL←0

对*于排序中的j（I_k，上升顺序由x_{jk}）确定*
GL←GL + gj，HL←HL + hj GR ←G-GL，HR←H - HL
得分←max（得分， $\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}$ *）*

结束

//列举遗漏值左转
$G_R \leftarrow 0$，HR ← 0
对*于排序中的j（I_k，下降顺序x_{jk}）*
GR←GR + gj，HR←HR + hj GL ←G - GR，HL←H - HR
得分←max（得分， $\frac{G_L^2}{H_L+\lambda} + \frac{G_R}{H_R+\lambda} - \frac{G^2}{H+\lambda}$ *）*

结束

输出： 具有最大增益的分割和默认方向

分为默认方向。 每个分支有两种默认方向选择。 从数据中学习最佳的默认方向。 该算法显示在Alg中。3。关键的改进是只访问非缺失条目I_k。 所提出的算法将不存在作为缺失值处理，并学习处理缺失值的最佳方向。通过将枚举限制为一致的解决方案，当不存在对应于用户指定的值时，也可以应用相同的算法。

据我们所知，大多数现有的树形学习算法或者仅针对密集数据进行了优化，或者需要特定的程序来处理有限的情况，例如分类编码。 XGBoost以统一的方式处理所有稀疏模式。 更重要的是，我们的方法利用稀疏性使计算复杂度与输入中非缺失条目的数量成线性关系。 图。5显示了稀疏感知和在一个Allstate-10K数据集上的幼稚实现的比较（第二部分给出的数据集的描述）。6）。 我们发现稀疏感知算法的运行速度比原始版本快50倍。 这证

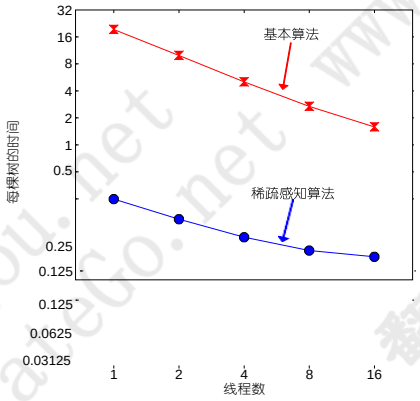


图5：稀疏感知算法对Allstate-10K的影响。 数据集稀疏主要是由于单编码。 稀疏感知算法

比天真版快50倍以上
这并不考虑稀疏性。

4. 系统设计

4.1 并行学习的列块

树学习中最耗时的部分是获得

数据分类排序。 为了降低成本

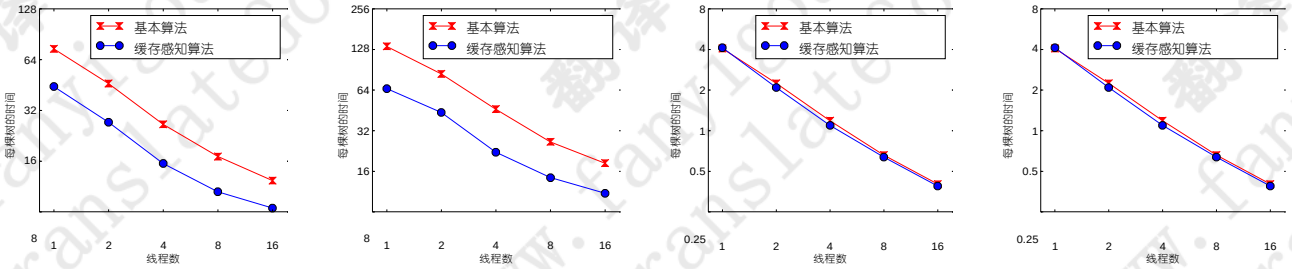
我们建议将数据存储在内存单元中，我们称之为块。 每个块中的数据存储在

实了稀疏感知算法的重要性。

压缩列（CSC）格式，每列都有排序由相应的特征值。 这个输入数据布局只需要在训练之前计算一次，并且可以在以后的迭代中重复使用。

在精确的贪婪算法中，我们将整个数据集存储在一个块中，并通过对预先排序的条目进行线性扫描来运行分割搜索算法。我们对所有叶子进行分组查找，因此对块的一次扫描将收集所有叶子分支中的分割候选者的统计数据。 图。6 展示了我们如何将数据集转换为格式并使用块结构找到最佳分割。 块结构也有助于使用近似算法。 在这种情况下可以使用多个块，每个块对应于数据集中行的子集。 不同的块可以分布在不同的机器上，也可以在磁盘外存储设置中存储。 使用排序后的结构，分位数查找步骤变为对已排序列进行线性扫描。 这对于本地提议算法尤其有用，在这些算法中，每个分支频繁地生成候选项。直方图聚合中的二进制搜索也成为线性时间合并样式算法。

为每个列收集统计数据可以并行化，为我们提供了一个分裂查找的并行算法。 重要的是，列块结构也支持列子采样，因为在块中选择列的子集很容易。



(a) Allstate 10M (b) 希格斯10M (c) Allstate 1M (d) Higgs 1M

图7：高速缓存感知预取在精确贪婪算法中的影响。 我们发现cache-miss效应会影响大型数据集（一千万个实例）的性能。 当数据集很大时，使用缓存感知预取可将性能提高一倍。

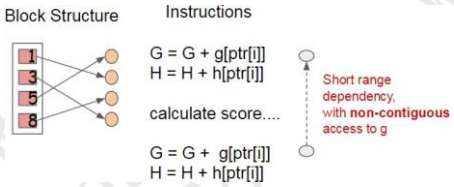


图8： 短程数据依赖模式由于缓存未命中可能导致停顿。

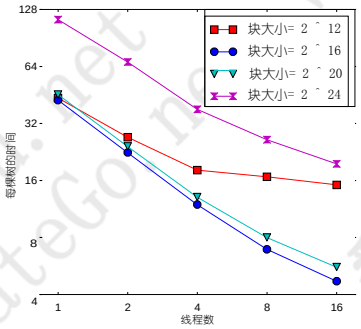
时间复杂度分析设d是树的最大深度，K是树的总数。对于精确贪婪算法，原始空间感知算法的时间复杂度为0（Kd x 0 log n）。 这里我们使用x 0来表示训练数据中非缺失条目的数量。 另一方面，在块结构上的树推进 –
成本0（Kd x 0 + x 0 log n）。 这里0（x 0 log n）是可以摊销的一次预处理成本。 该分析表明，块结构有助于节省额外的log n因子，这在n很大时很重要。对于近似算法，原始的时间复杂度
二元搜索算法是0（Kd x 0 log q）。这里q是数据集中提议候选人的数量。 虽然q通常在32到100之间，但对数因子仍然会引入开销。 使用块结构，我们可以将时间减少到0（Kd x 0 + x 0 log B），其中B是每个块中的最大行数。 我们再次可以保存附加信息

计算中记录q因子。

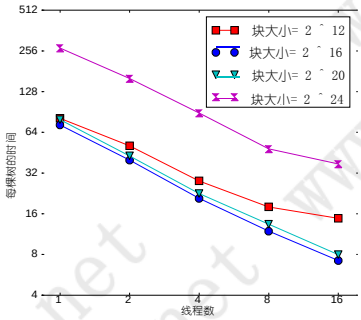
4.2 缓存感知访问

尽管所提出的块结构有助于优化分裂查找的计算复杂性，但新算法需要通过行索引间接获取梯度统计信息，因为这些值是按特征顺序访问的。 这是一个非连续的内存访问。 分裂枚举的一个简单实现引入了累积和非连续内存获取操作之间的即时读/写依赖性（见图2）。 8）。 当渐变统计信息不适合CPU缓存并发生缓存未命中时，这会减慢拆分查找速度。

对于精确的贪婪算法，我们可以通过缓存感知预取算法来缓解这个问题。 具体来说，我们在每个线程中分配一个内部缓冲区，将梯度统计信息提取到它中，然后以小批量方式执行累加。 此预取将直接读/写依赖关系更改为更长的依赖关系，并有助于在行数很大时减少运行时开销。数字7 给出了缓存感知与比较



(a) Allstate 10M



(b) 希格斯10M

图9：块大小对近似算法的影响。 我们发现过小的块会导致低效的并行化，而过大的块也会由于缓存未命中而减慢训练速度。

Higgs和Allstate数据集中的非缓存感知算法。 我们发现，当数据集很大时，精确贪婪算法的缓存感知实现的运行速度比原始版本快两倍。

对于近似算法，我们通过选择正确的块大小来解决问题。 我们将块大小定义为包含在块中的最大示例数，因为这反映了梯度统计信息的高速缓存存储成本。 选择过小的块大小会导致每个线程的工作量很小，并导致低效的并行化。 另一方面，过大的块导致缓存未命中，因为梯度统计不适合CPU缓存。 块大小的一个很好的选择平衡这两个因素。 我们在两个数据集上比较了块大小的各种选择。 结果在图2中给出。9。 这个结果验证了我们的讨论和

表1：主要树木增效系统的比较。

系统	精确 贪婪	近似 全球	近似 本地	外的核心	稀疏 知道的	平行
XGBoost	是	是	是	是	是	是
pGBRT	没有	没有	是	没有	没有	是
Spark MLLib	没有	是	没有	没有	部分	是
H2O	没有	是	没有	没有	部分	是
scikit学习	是	没有	没有	没有	没有	没有
R GBM	是	没有	没有	没有	部分	没有

显示每个块选择2个\示例均衡缓存属性和并行化。

4.3 用于核心外计算的块

我们的系统的一个目标是充分利用机器的资源来实现可扩展的学习。 除了处理器和内存外，利用磁盘空间处理不适合主内存的数据非常重要。 为了启用核心外计算，我们将数据分成多个块并将每个块存储在磁盘上。 在计算过程中，使用独立的线程将块预取到主存储缓冲区中非常重要，因此可以在读取磁盘的同时进行计算。 但是，这并不能完全解决问题，因为磁盘读取占用了大部分计算时间。 减少开销并增加磁盘IO的吞吐量非常重要。 我们主要使用两种技术来改善核心外计算。

块压缩我们使用的第一种技术是块压缩。 该块由列压缩，并在加载到主内存时由独立线程进行解压缩。 这有助于将部分解压缩计算与磁盘读取成本进行交易。 我们使用通用压缩算法来压缩特征值。 对于行索引，我们将行索引减去块的开始索引，并使用16位整数来存储每个偏移量。 这需要每块2个\示例，这被证实是一个好的设置。 在我们测试的大部分数据集中，我们实现了大约26%到29%的压缩比。

块分片第二种技术是以另一种方式将数据分片到多个磁盘上。 预取线程分配给每个磁盘并将数据提取到内存缓冲区中。 训练线程然后交替地从每个缓冲区中读取数据。 这有助于在多个磁盘可用时提高磁盘读取的吞吐量。

5. 相关作品

我们的系统实现梯度提升[10]，它在功能空间中执行加法优化。 梯度树提升已成功用于分类[12]，学习排名[5]，结构化预测[8]以及其他领域。 XGBoost采用正则化模型来防止过度配合。 这与以前在正规化的贪婪森林中的工作相似[25]，但简化了并行化的目标和算法。列抽样是从RandomForest借来的一种简单而有效的技术[4]。 稀疏意识学习在其他类型的模型中至关重要，例如

线性模型[9]，很少有关于树木学习的着作考虑过这个话题在原则上。 本文提出的算法是第一个处理各种稀疏模式的统一方法。

并行化树学习有几个已有的工作[22, 19]。 大多数这些算法都属于本文所述的近似框架。 值得注意的是，也可以按列分区数据[23]并应用精确的贪婪算法。 这在我们的框架中也得到了支持，并且可以使用诸如缓存感知预检技术来使这种类型的算法受益。 尽管大多数现有着作都集中在并行化的算法方面，但我们的工作改进了两个未探索的系统方向：非核心计算和高速缓存感知学习。 这为我们提供了有关系统和算法如何进行联合优化的见解，并提供了一个端对端系统，可以利用非常有限的计算资源处理大规模问题。 我们还总结了我们的系统和Table中现有的开源实现之间的比较1。 分位总结（无权重）是数据库社区中的一个经典问题[14, 24]。 然而，近似树推进算法揭示了一个更加普遍的问题 – 在加权数据上发现分位数。 据我们所知，本文提出的加权分位图是解决这个问题的第一种方法。 加权分位总结也并非特定于树学习，并且可以使数据科学中的其他应用受益并在未来机器学习。

6. 结束评估

6.1 系统实施

我们将XGBoost作为开源软件包实施[6]。 该包是便携式和可重复使用的。 它支持各种加权分类和等级目标函数，以及用户定义的目标函数。 它可以使用python，R，Julia等流行语言，并与语言原生数据科学流水线（如scikit-learn）自然结合。 分布式版本建立在rabit库之上[7]为allreduce。 XGBoost的可移植性使其在许多生态系统中可用，而不是仅限于特定的平台。 分布式XGBoost本身在Hadoop，MPI Sun Grid引擎上运行。 最近，我们还在jvm bigdata堆栈上启用了分布式XGBoost，如Flink和Spark。 分布式版本也已经整合到云平台天池[8]的阿里巴巴。 我们相信未来会有更多的整合。

6.2 数据集和设置

我们在实验中使用了四个数据集。 表中列出了这些数据集的摘要2。 在一些实验中，

[6] https://github.com/dmlc/xgboost \https://github.com/dmlc/rabit\https://tianchi.aliyun.com

表2：实验中使用的数据集。

数据集	n	m	任务
好事达	10 M	4227	保险索赔分类
希格斯玻色子	10 M	28	事件分类
雅虎LTRC	473K	700	学习排名
Criteo	1.7 B	67	点击率预测

我们使用一个随机选择的数据子集，这可能是由于基线较慢，或者用不同的数据集大小来演示算法的性能。

我们使用后缀来表示这些情况下的大小。 例如，Allstate-10K意味着包含10K实例的Allstate数据集的一个子集。 我们使用的第一个数据集是Allstate保险索赔数据集[9]。 其任务是预测给定不同风险因素的保险索赔的可能性和成本。 在实验中，我们将任务简化为仅预测保险索赔的可能性。 该数据集用于评估第2节中稀疏感知算法的影响。3.4。 这些数据中的大部分稀疏特征来自单热编码。 我们随机选择10M实例作为训练集并使用它休息作为评估集。

第二个数据集是希格斯玻色子数据集[10]来自高能物理。 数据是利用物理事件的蒙特卡罗模拟产生的。 它包含21个由加速器中的粒子探测器测量的运动特性。 它还包含七个额外的派生物理量的粒子。 任务是分类一个事件是否对应于希格斯玻色子。 我们随机选择10M个实例作为训练集，并将其余的作为评估集。 第三个数据集是Yahoo！学习排名挑战数据集[6]，这是学习排名算法最常用的基准之一。 该数据集包含20K个网页搜索查询，每个查询对应大约22个文档的列表。 任务是根据查询的相关性对文档进行排名。 我们使用官方列车测试分裂在我们的实验。

最后一个数据集是criteo terabyte点击日志数据集[11]。我们使用这个数据集来评估系统在核外和分布式设置中的缩放属性。 数据包含13个整数特征和26个用户，物品和广告商信息的ID特征。 由于基于树的模型在处理连续特征方面更好，我们通过计算前10天的ID特征的平均点击率和计数的统计数据来预处理数据，在接下来的十天内将ID特征替换为相应的统计数据以用于训练。预处理后的训练集包含17个具有67个特征的实例（13个整数，26个平均CTR统计量和26个计数）。 整个数据集在LibSVM格式中超过一兆字节。

我们将前三个数据集用于单机并行设置，并将最后一个数据集用于分布式和非内核设置。 所有的单机实验都是在戴尔PowerEdge R420上搭配两枚八核Intel Xeon（E5-2470）（2.3GHz）和64GB内存进行的。 如果未指定，则使用机器中所有可用的核心运行所有实验。 将在下面描述分布式和外核实验的机器设置

\的https://www.kaggle.com/c/ClaimPredictionChallenge
\https://archive.ics.uci.edu/ml/datasets/HIGGS\http://labs.criteo.com/downloads/download-terabyte-点击日志/

表3：在Higgs-1M数据上的具有500棵树的精确贪婪方法的比较。

方法	每棵树的时间（秒）	测试AUC
XGBoost	0.6841	0.8304
XGBoost (colsample = 0.5)	0.6401	0.8245
scikit学习	28.51	0.8302
R.gbm	1.032	0.6224

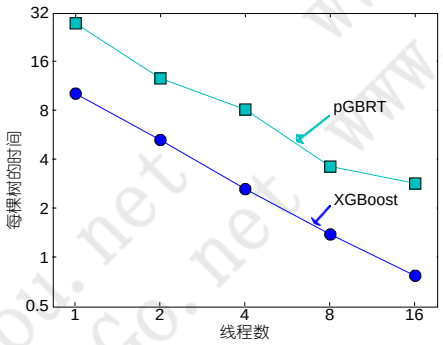


图10：雅虎LTRC数据集上的XGBoost和pG-BRT之间的比较。

表4：雅虎500级树学习排名比较 LTRC数据集

方法	每棵树的时间（秒）	NDCG@10
XGBoost	0.826	0.7892
XGBoost (colsample = 0.5)	0.506	0.7913
pGBRT [22]	2.576	0.7915

相应的部分。 在所有的实验中，我们使用最大深度等于8的常用设置来增强树木，收缩率等于0.1，并且除非明确指定，否则不进行列子采样。 当我们使用其他最大深度设置时，我们可以找到类似的结果。

6.3 分类

在本节中，我们通过将Higgs-1M数据上的精确贪婪算法与其他两种常用的精确贪婪树增强实现进行比较，评估XGBoost在单机上的性能。 由于scikit-learn仅处理非稀疏输入，我们选择密集的Higgs数据集进行公平比较。 我们使用1M子集使scikit-learn在合理的时间内完成运行。 在比较的方法中，R的GBM使用贪婪的方法，只扩展树的一个分支，这使得它更快，但可能导致精度较低，而scikit-learn和XGBoost都学习了一棵完整的树。 结果显示在表中3。 XGBoost和scikit-learn都比R的GBM提供更好的性能，而XGBoost的运行速度比scikit-learn快10倍以上。 在这个实验中，我们还发现列子样本的性能比使用所有特征的性能要差一些。这可能是由于这个数据集中很少有重要特征，我们可以从所有特征中进行贪婪选择。

6.4 学习排名

接下来我们评估XGBoost在学习排序问题上的表现。我们比较pGBRT [22]，这是此前任务中最好的公开系统。XGBoost

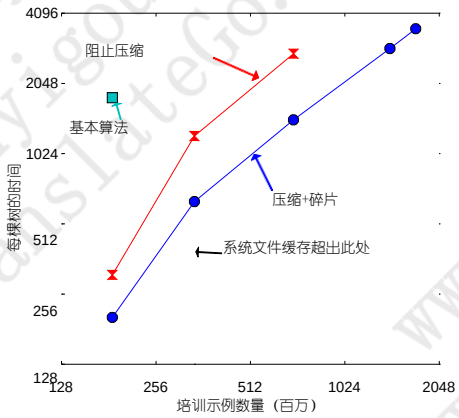


图11：criteo数据的不同子集上的外核方法比较。缺少的数据点是由于磁盘空间不足造成的。 我们可以发现基本算法只能处理200M的例子。 添加压缩功能可以提供3倍的加速，并且分成两块磁盘可以提供2倍的加速。 系统耗尽文件缓存从400M开始实例。 在这之后，算法真的必须依靠磁盘。 压缩+分片方法在用完文件缓存时速度较慢，并在之后呈现线性趋势。

运行精确的贪婪算法，而pGBRT只支持近似算法。 结果显示在表中4 和图。10。 我们发现XGBoost运行速度更快。 有趣的是，子采样列不仅缩短了运行时间，而且还为这个问题提供了更高的性能。 这可能是由于子采样有助于防止过度配合，这是许多用户所观察到的。

6.5 非核心实验

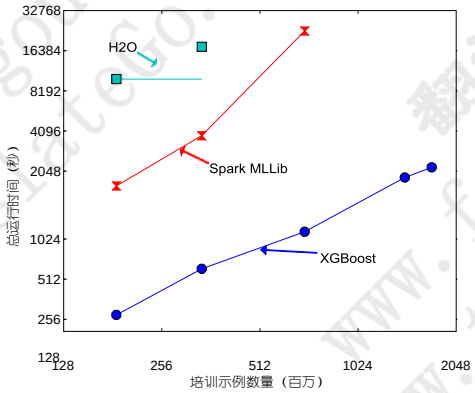
我们还在criteo数据的核心外设置中评估我们的系统。我们在一台AWS c3.8xlarge机器上（32个核心，两个320 GB SSD，60 GB RAM）进行了实验。 结果如图所示11。 我们可以发现压缩有助于将计算速度提高三倍，分成两块磁盘进一步提供2x加速。 对于这种类型的实验，使用非常大的数据集来排空系统文件缓存以实现真正的核心外设置非常重要。 这确实是我们的设置。 当系统用完文件缓存时，我们可以观察到一个转换点。 请注意，最终方法中的转换并不那么显着。 这是由于更大的磁盘吞吐量和更好的计算资源利用率。 我们的最终方法是可以处理的
一台机器上有17亿个例子。

6.6 分布式实验

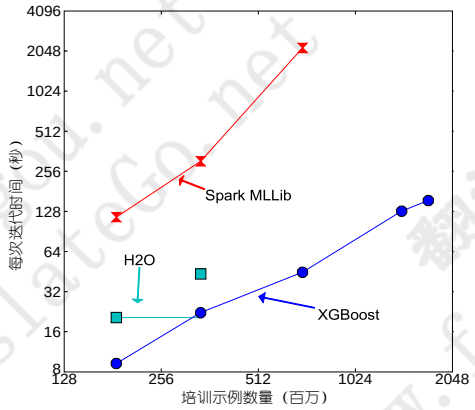
最后，我们在分布式设置中评估系统。 我们在EC2上搭建了一个YARN集群，配备了m3.2xlarge机器，这对于集群来说是非常普遍的选择。 每台机器包含8个虚拟内核，30GB RAM和2个80GB SSD本地磁盘。 数据集存储在AWS S3而不是HDFS上，以避免购买持久性存储。

我们首先将我们的系统与两个生产级别的分布式系统进行比较：Spark MLLib [18]和H2O[12]。 我们使用32 m3.2xlarge机器并测试系统的性能，

[12]www.h2o.ai



(a) 端到端时间成本包括数据加载



(b) 每次迭代成本不包括数据加载

图12：32个EC2节点上的不同分布式系统在criteo数据的不同子集上进行10次迭代的比较。XGBoost的运行速度比每次迭代的10倍还要多，而H2O的优化版本的运行速度是2.2x（但是，H2O在加载数据时速度较慢，端到端时间变得更糟）。 请注意，当内存不足时，spark会急剧减速。 通过利用核心外计算，XGBoost运行速度更快，可以在给定资源的情况下平稳扩展到全部17亿个示例。

与各种输入大小的tems。 这两个基准系统都是内存分析框架，需要将数据存储在RAM中，而XGBoost可以在内存不足时切换到外核设置。 结果如图2所示。12。 我们可以发现XGBoost运行速度比基准系统快。 更重要的是，它能够利用核心外计算，并利用给定的有限计算资源平滑扩展到全部17亿个示例。 基准系统只能用给定资源处理数据的子集。 该实验显示了将所有系统改进结合在一起并解决实际规模问题的优势。 我们还通过改变机器数量来评估XGBoost的缩放属性。 结果如图2所示。13。 我们可以发现XGBoost的性能随着我们添加更多机器而线性扩展。 重要的是，XGBoost能够处理整个1.7 十亿只数据只有四台机器。 这显示了系统处理更大数据的潜力。

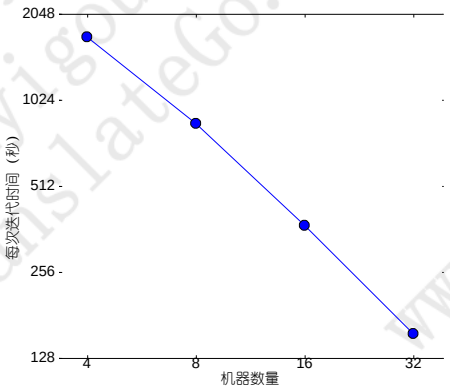


图13：使用不同数量的机器在criteo满17亿个数据集上扩展XGBoost。 使用更多的机器会产生更多的文件缓存，并使系统运行得更快，从而使趋势略微超线性。 XGBoost可以使用四台机器处理整个数据集，并通过利用更多可用资源来平滑扩展。

7. 结论

在本文中，我们描述了我们在构建XGBoost时学到的经验教训，XGBoost是一种可扩展的树型增强系统，被数据科学家广泛使用，并在许多问题上提供了最新的结果。 我们提出了一种用于处理稀疏数据的新颖的稀疏感知算法和一种用于近似学习的理论上合理的加权分位图。 我们的经验表明，高速缓存访问模式，数据压缩和分片是构建可扩展的树木增效端到端系统的基本要素。这些经验教训也可以应用于其他机器学习系统。 通过结合这些见解，XGBoost能够使用最少的资源解决真实世界的规模问题。

致谢

我们要感谢Tyler B. Johnson，Marco Tulio Ribeiro，Sameer Singh，Arvind Krishnamurthy的宝贵意见。 我们也衷心感谢Tong He，Bing Xu，Michael Benesty，Yuan Tang，Hongliang Liu，Qiang Kou，Nan Zhu等XGBoost社区的贡献者。 这项工作部分得到了ONR（PECASE）N000141010672，NSF IIS 1258741的支持
以及由MARCO和DARPA赞助的TerraSwarm研究中心。

8. 参考

[1] R. Bekkerman。 kdd杯比赛的现状和未来：一个局外人的视角。

[2] R. Bekkerman，M. Bilenko和J. Langford。 扩大机器学习：并行和分布式方法。 剑桥大学出版社，纽约，纽约，美国，2011年。

[3] J. Bennett和S.兰宁。 netflix奖。 在KDD Cup Workshop 2007会议论文集，第3-6页，纽约，2007年8月。

[4] L. Breiman。 随机森林。 Maching Learning，45（1）：5-32，2001年10月。

[5] C.布尔曰。 从ranknet到lambdarank到lambdamart：概述。 学习，11：23-581,2010。

[6] O. Chapelle和Y. Chang。 雅虎 学习挑战挑战概述。 机器学习研究杂志 - W&CP，14：1-24，2011。

[7] T. Chen，H. Li，Q. Yang和Y. Yu。 一般功能矩阵分解使用梯度提升。 正在进行*第30届国际机器学习会议（ICML'13），第1卷，第436-444页，2013年。*

[8] T. Chen，S. Singh，B. Taskar和C. Guestrin。 针对条件随机场的高效二阶梯度提升。 第18届人工智能与统计学会议（AISTATS'15）第1卷，2015年。

[9] 回覆。 范，K.-W. Chang，C.-J. Hsieh，X.-R.王和C.-J.林。 LIBLINEAR：用于大型线性分类的库。 Journal of Machine Learning Research，9：1871-1874,2008。

[10] J.弗里德曼。 贪心功能逼近：梯度提升机器。 统计学年报，29（5）：1189-1232，2001。

[11] J.弗里德曼。 随机梯度提升。 Computational Statistics&Data Analysis，38（4）：367-378,2002。

[12] J. Friedman，T. Hastie和R. Tibshirani。 添加剂物流回归：提升的统计视图。 Annals of Statistics，28（2）：337-407,2000。

[13] JH弗里德曼和BE Popescu。 重要抽样学习合奏，2003年。

[14] M. Greenwald和S. Khanna。 分位数摘要的空间高效在线计算。 在2001年ACM SIGMOD国际数据管理会议论文集，第58-66页，2001年。

[15] 他，J. Pan，O. Jin，T. Xu，B. Liu，T. Xu，Y. Shi，A. Atallah，R. Herbrich，S. Bowers和JQ n。 坎德拉。 预测在Facebook上点击广告的实用经验。 在第八届网络广告数据挖掘国际研讨会论文集中，ADKDD'14,2014。

[16] P. *Li。 强大的Logitboost和自适应基类（ABC）Logitboost。 在第二十六届会议人工智能不确定性年会（UAI'10），第302-311页，2010年。*

[17] P. Li，Q. Wu和CJ Burges。 Mcrank：学习使用多重分类和渐变增强进行排名。 在Advances in Neural Information Processing Systems 20，pages 897-904。2008年。

[18] X. Meng，J.Bradley，B.Yavuz，E.Sparks，S. Venkataraman，D. Liu，J. Freeman，D. Tsai，M. Amde，S. Owen，D. Xin，R. Xin，MJ Franklin，R. Zadeh，M. Zaharia和A. Talwalkar。 MLlib：在apache中进行机器学习。 机器学习研究杂志，17（34）：1-7，2016。

[19] B. Panda，JS Herbach，S. Basu和RJ Bayardo。 星球：用mapreduce大量并行学习树合奏。 VLDB Endowment，2（2）：1426-1437，2009年8月。

[20] F. Pedregosa，G. Varoquaux，A. Gramfort，V. Michel，B. Thirion，O. Grisel，M. Blondel，P. Prettenhofer，R. Weiss，V. Dubourg，J. Vanderplas，A. Passos，D. Cournapeau，M.Brucher，M.Perrot和E.Duchesnay。 Scikit-learn：Python中的机器学习。 Journal of Machine Learning Research，12：2825-2830，2011。

[21] G. *Ridgeway。 广义增强模型：gbm包的指南。*

[22] S. Tyree，K. Weinberger，K. Agrawal和J. Paykin。 为web搜索排名并行推动回归树。 在第20届万维网国际会议论文集，第387-396页。 ACM，2011。

[23] J. Ye，J.-H. Chow，J. Chen和Z. Zheng。 随机梯度推动分布式决策树。 在第18届ACM信息和知识管理会议论文集中，CIKM'09。

[24] Q. *Zhang和W. Wang。 高速数据流近似分位数的快速算法。在第19届国际科学和统计数据库管理会议论文集中，2007年。*

[25] T. Zhang和R. Johnson。 利用正则化贪婪森林学习非线性函数。 IEEE Transactions on Pattern Analysis and Machine Intelligence，36（5），2014。