

XGBoost解析系列-原理

原创 cyber19 最后发布于2017-11-26 00:01:10 阅读数 5888 ☆ 收藏

👍
10

🔖

💬
2

📖

☆

📱

<

>

💰

前言

Boosting算法框架

XGBoost原理推导

XGBoost算法

XGBoost工程优化

XGBoost算法复杂度

参考资料

0.前言

解析源码之前，还是介绍说明下XGBoost原理，网上对于XGBoost原理已有各种版本的解读。而这篇博客，笔者主要想根据自己的理解，梳理看过的资料，包括陈天奇的论文以及引用论文内容，本文主要内容基于陈天奇的论文与PPT，希望能够做到系统地介绍XGBoost，同时加入源码新特性让内容

XGBoost不仅能在单机上通过OMP实现高度并行化，还能通过MPI接口与近似分位点算法（论文中是weighted quantiles sketch）实现高效的分布式近似分位点算法（approximate quantiles）会附加一篇博客进行详细说明，分位点算法在分布式系统、流式系统中真的是个很天才的想法，很多基石。最早由M.Greenwald和S. Khanna与2001年提出的GK Summay算法，直到2007年被Q. Zhang和W. Wang提出的多层level的merge与com框架进行高度优化，而被称为A fast algorithm for approximate quantiles，详情见下一篇博客。

1.Boosting算法框架

XGBoost算法属于集成学习中的boosting分支，其算法框架遵循1999年Friedman提出的boosting框架，该分支还有GBDT(Gradient Boosting Tree)，boosting集成是后一个模型是对前一个模型产生误差信息进行矫正。gradient boost更具体，新模型的引入是为了减少上个模型的残差(residual)在残差减少的梯度(Gradient)方向上建立一个新的模型。Friedman论文中针对回归过程提出boost框架如下：

| | |
|---|--|
| | Algorithm 1: Gradient Boost |
| 1 | $F_0(\mathbf{x}) = \arg \min_{\rho} \sum_{i=1}^N L(y_i, \rho)$ |
| 2 | For $m = 1$ to M do: |
| 3 | $\tilde{y}_i = - \left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)} \right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, i = 1, N$ |
| 4 | $\mathbf{a}_m = \arg \min_{\mathbf{a}, \beta} \sum_{i=1}^N [\tilde{y}_i - \beta h(\mathbf{x}_i; \mathbf{a})]^2$ |
| 5 | $\rho_m = \arg \min_{\rho} \sum_{i=1}^N L(y_i, F_{m-1}(\mathbf{x}_i) + \rho h(\mathbf{x}_i; \mathbf{a}_m))$ |
| 6 | $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \rho_m h(\mathbf{x}; \mathbf{a}_m)$ |
| 7 | endFor |
| | end Algorithm |

<http://blog.csdn.net>

Friedman提出boost算法框架过程描述如下：

1. 设定函数初始值 F_0 ，为一个恒值函数，论文中基于变量优化出恒值，实际上也可以给定任意值或者直接为0。
2. 根据参数 M ，进行 M 次迭代，不断将当前函数 F_{m-1} 往最优函数 F^* 空间上逼近，逼近方向就是当前函数下的函数负梯度方向 $-\nabla L(y, F)|_{F=F_{m-1}}$ ，而非变量，本质上属于泛函优化。
3. 每次迭代计算出函数负梯度，基于训练数据构建模型来拟合负梯度。原则上可以选择任何模型：树模型，线性模型或者神经网络等等，很少框架推测：神经网络容易过拟合，后续函数负梯度恒为0就无法继续迭代优化下去。如果用树模型进行拟合，就是我们熟悉的CART模型。
4. 优化步长，根据目标函数来最优步长 ρ_m ，属于变量优化，并更新当前函数，继续迭代。框架并没有shrinkage机制来控制步长，采用树模型可能过度拟合，目前现代的boosting框架都支持shrinkage，即最终的优化步长应乘以shrinkage参数： $\rho_m = \rho_m \gamma$ 。

该框架实际上是泛函梯度下降优化过程，尽管中间局部包含变量优化步骤，对比变量优化迭代不难发现相似之处。准确来说变量优化的其他算法优化：1) 基于梯度下降优化，步长优化可以是精确优化和非精确优化。2) 基于牛顿法，根据二阶梯度直接计算步长 $f''(x)^{-1}$ ，即更新变量

谈到集成学习，不得不说bagging集成，比如随机森林，1）建树前对样本随机抽样（行采样），2）每个特征分裂随机采样（列采样），3）每个特征分裂随机选择分裂点（阈值采样），4）每个特征分裂随机选择分裂方向（左右子节点选择）。由于每个特征分裂都是随机采样，因此能够避免过拟合。Python机器学习包sklearn中随机森林RF能完全并行训练，而GBDT算法不行，训练过程还是单线程，无法利用多核导致速度慢。后续优化实现并行不是同时构造N颗树，而是单颗树构建中遍历最优特征时的并行，类似XGBoost实现过程。随机森林中行采样与列采样有效避免过拟合，XGBoost通过正则化项避免过拟合，此外其还支持Dropout抗过拟合。

2. XGBoost原理推导

1. XGBoost考虑正则化项，目标函数定义如下：

$$L(\phi) = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k), \text{ 其中 } \Omega(f_k) = \gamma T + \frac{1}{2} \lambda \|w\|^2$$

其中 \hat{y}_i 为预测输出， y_i 为label值， f_k 为第 k 棵树模型， T 为树叶子节点数， w 为叶子权重值， γ 为叶子树惩罚正则项，具有防止过拟合的作用， λ 为叶子权重正则项，防止过拟合。XGBoost也支持一阶正则化，容易优化叶子节点权重为0，不过不常用。

根据Boosting框架，可以优化出树的建模函数 $f_t(x)$ ：

$$\begin{aligned} L^{(t)} &= \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \\ &\approx \sum_{i=1}^n [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_t f_t^2(x_i)] + \Omega(f_t) \\ &= \sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_t f_t^2(x_i)] + \Omega(f_t) + \text{constant} \end{aligned}$$

2. 因此，每次建树优化以下目标：

$$\hat{L}^{(t)} = \sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_t f_t^2(x_i)] + \Omega(f_t)$$

其中 $g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$ ， $h_i = \partial_{\hat{y}_i^{(t-1)}}^2 l(y_i, \hat{y}_i^{(t-1)})$ ，而且：

$$\Omega(f_t) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$

3. 假设我们已知树结构 q ，即每个样本 x_i 能通过该结构 q 找到对应的叶子节点 j ，可以定义 $I_j = \{i | q(x_i) = j\}$ 为在树结构 q 下，落入叶子节点 j 的集合。展开上述表达式并通过配方法不难得到：

$$\begin{aligned} \hat{L}^{(t)} &= \sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_t f_t^2(x_i)] + \Omega(f_t) \\ &= \sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_t f_t^2(x_i)] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 \\ &= \sum_{j=1}^T [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T \\ &= \frac{1}{2} \sum_{j=1}^T (H_j + \lambda) (w_j + \frac{G_j}{H_j + \lambda})^2 + \gamma T - \frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} \end{aligned}$$

其中 $G_j = \sum_{i \in I_j} g_i$ 为落入叶子 i 所有样本一阶梯度统计值总和， $H_j = \sum_{i \in I_j} h_i$ 为落入叶子 i 所有样本二阶梯度统计值总和。最终得到叶子权重值为：

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

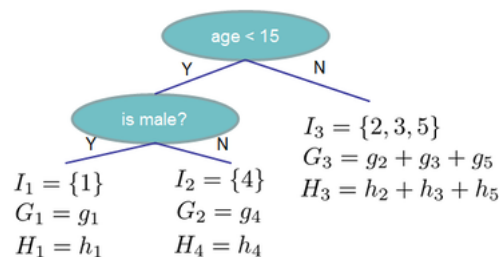
4. 最终的目标值为：

$$\hat{L}^* = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

下图为树的目标值计算样例：



| 样本号 | 梯度数据 |
|-----|------------|
| 1 | g_1, h_1 |
| 2 | g_2, h_2 |
| 3 | g_3, h_3 |
| 4 | g_4, h_4 |
| 5 | g_5, h_5 |



$$Obj = -\frac{1}{2} \sum_j \frac{G_j^2}{H_j + \lambda} + 3\gamma$$

这个分数越小，代表这个树的结构越好

http://blog.csdn.net/matrix_zz1

10

2

2

2

2

2

2

2

2

2

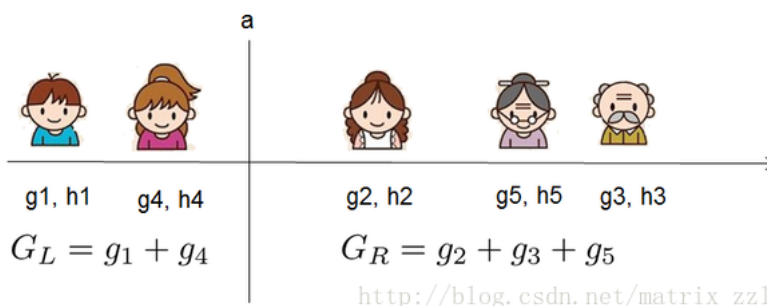
5. 回顾步骤3，可以发现前提假设是已知树结构 q ，除非遍历所有树结构，否则无法优化最优目标值，而且为了优化目标值，我们也不可能遍历所有树结构。论文提出了类似于CART定义增益公式来启发式的寻找最优树结构，若当前树结构 I 能被分裂成 I_L 与 I_R ， $I = I_L \cup I_R$ ，XGBoost的增益公式：

$$L_{split} = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$

3. XGBoost算法

1) XGBoost精确贪婪算法

构建树流程如下：1. 遍历每个特征 k ，2) 遍历当前特征 k 下每个取值 x_{jk} ，对于特征分裂值将前节点样本样本划分到左右子树，根据上述公式通过取增益最大对应的特征以及特征分裂值，执行节点分裂， L_{split} 最大值小于0则停止分裂， γ 可以视为分裂阈值，起到一定程度的预剪枝的作用，再不断根据特征值排序，从左到右进行扫描来找出当前特征下最优分裂值。



http://blog.csdn.net/matrix_zz1

论文提出的精确贪婪算法流程如下：

2

2

2

Algorithm 1: Exact Greedy Algorithm for Split Finding**Input:** I , instance set of current node**Input:** d , feature dimension $gain \leftarrow 0$ $G \leftarrow \sum_{i \in I} g_i, H \leftarrow \sum_{i \in I} h_i$ **for** $k = 1$ **to** m **do** $G_L \leftarrow 0, H_L \leftarrow 0$ **for** j **in** $sorted(I, \text{by } \mathbf{x}_{jk})$ **do** $G_L \leftarrow G_L + g_j, H_L \leftarrow H_L + h_j$ $G_R \leftarrow G - G_L, H_R \leftarrow H - H_L$ $score \leftarrow \max(score, \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda})$ **end****end****Output:** Split with max score

10



2



赏

https://blog.csdn.net/matrix_zz1

2) XGBoost近似算法

精确算法由于需要遍历特征的所有取值，计算效率低，适合单机小数据，对于大数据、分布式场景并不适合。论文基于Weighted Quantile Sketec提出相应的近似算法，也证明了该分位点的正确性。通过设置 ϵ 来设置近似程度，而且论文给出近似算法的2种方案：

1. 在建树之前预先将数据进行全局分桶，需要设置更小的 ϵ ，产生更多的桶，特征分裂查找基于候选点多，计算较慢，但只需在全局执行一次。
2. 每次分裂重新局部分桶，可以设置较大的 ϵ ，产生更少的桶，每次特征分裂查找基于候选点少，计算速度快，但是需要每次节点分裂后重新执行方案更适合树深的场景。

论文给出Higgs案例下，方案1全局分桶设置 $\epsilon = 0.05$ 与精确算法效果差不多，方案2局部分桶设置 $\epsilon = 0.3$ 与精确算法仅稍差点，方案1全局分桶 $\epsilon = 0.3$ 则效果极差。

近似算法为什么能用于分布式？主要原因是分桶是基于分位点算法，分位点算法支持merge和prune操作，**想了解该过程可以移步《分位点算法详XGBoost场景属于weighted分位点算法**，作者在论文后面也证明weighted分位点算法支持merge和prune操作，因此适合与分布式场景。近似算法主进行分桶，同时希望每个桶尽量均匀。考虑数据集：

$$D_k = \{(x_{1k}, h_1), (x_{2k}, h_2), \dots, (x_{nk}, h_n)\}$$

定义rank函数为 $r_k : R \rightarrow [0, +\infty)$ ，二阶导数 h_i 一定大于等于0，而一阶导数 g_i 则不具备该条件，所以无法构建分位点。实际上XGBoost源代码不仅会构建 h_i 的分位行拆分，分别构建 $g_i > 0$ 集合分位点和 $g_i < 0$ 集合分位点（取负），目前按照论文中仅考虑二阶导数统计值 h_i ：

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

$r_k(z)$ 表示特征值小于 z 的样本集合中， h 累计值的百分占比。在这个排序函数下，我们找到一组点 $s_{k1}, s_{k2}, \dots, s_{kl}$ ，满足：

$$|r_k(s_{k,j}) - r_k(s_{k,j+1})| < \epsilon, \text{ 其中 } s_{k1} = \min_i x_{ik}, s_{kl} = \max_i x_{ik}$$

上述条件1为均匀条件，条件2为边界条件。这样就能得到 $1/\epsilon$ 个特征值分割候选点，假设数据量为1kw，设置 $\epsilon = 0.01$ ，则由候选点1kw降低为100，速度提升10w倍，精确贪婪算法流程如下：



举报





10



2

**Algorithm 2: Approximate Algorithm for Split Finding**

```
for  $k = 1$  to  $m$  do
    Propose  $S_k = \{s_{k1}, s_{k2}, \dots, s_{kl}\}$  by percentiles on feature  $k$ .
    Proposal can be done per tree (global), or per split(local).
end
for  $k = 1$  to  $m$  do
     $G_{kv} \leftarrow \sum_{j \in \{j | s_{k,v} \geq x_{jk} > s_{k,v-1}\}} g_j$ 
     $H_{kv} \leftarrow \sum_{j \in \{j | s_{k,v} \geq x_{jk} > s_{k,v-1}\}} h_j$ 
end
Follow same step as in previous section to find max
score only among proposed splits.
```

http://blog.csdn.net/matrix_zz1

3) XGBoost近似算法

对于数据缺失数据、one-hot编码等造成的特征稀疏现象，作者在论文中提出可以处理稀疏特征的分裂算法，主要是对稀疏特征值miss的样本学习分裂方向：

1. 默认miss value进右子树，对non-missing value的样本在左子树的统计值 G_L 与 H_L ，右子树为 $G - G_L$ 与 $H - H_L$ ，其中包含miss的样本。
 2. 默认miss value进左子树，对non-missing value的样本在右子树的统计值 G_R 与 H_R ，左子树为 $G - G_R$ 与 $H - H_R$ ，其中包含miss的样本。
- 最后，找出增益最大对于的特征、特征对于的值、以及miss value的分裂方向，作者在论文中提出基于稀疏分裂算法：

Algorithm 3: Sparsity-aware Split Finding

Input: I , instance set of current node

Input: $I_k = \{i \in I | x_{ik} \neq \text{missing}\}$

Input: d , feature dimension

Also applies to the approximate setting, only collect statistics of non-missing entries into buckets

$gain \leftarrow 0$

$G \leftarrow \sum_{i \in I} g_i, H \leftarrow \sum_{i \in I} h_i$

for $k = 1$ to m do

 // enumerate missing value goto right

$G_L \leftarrow 0, H_L \leftarrow 0$

 for j in sorted(I_k , ascent order by x_{jk}) do

$G_L \leftarrow G_L + g_j, H_L \leftarrow H_L + h_j$

$G_R \leftarrow G - G_L, H_R \leftarrow H - H_L$

$score \leftarrow \max(score, \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda})$

 end

 // enumerate missing value goto left

$G_R \leftarrow 0, H_R \leftarrow 0$

 for j in sorted(I_k , descent order by x_{jk}) do

$G_R \leftarrow G_R + g_j, H_R \leftarrow H_R + h_j$

$G_L \leftarrow G - G_R, H_L \leftarrow H - H_R$

$score \leftarrow \max(score, \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda})$

 end

end

Output: Split and default directions with max gain

http://blog.csdn.net/matrix_zz1

4. XGBoost工程优化

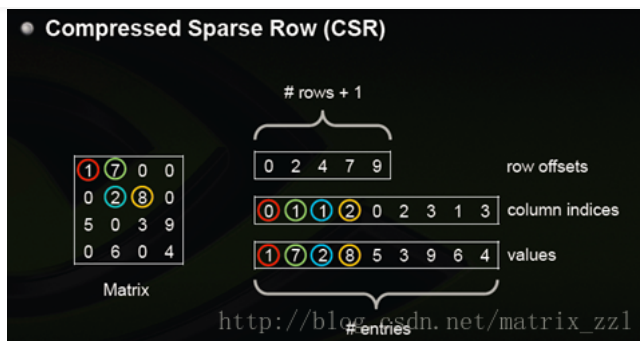
内部数据存储格式

从算法上看，每种算法都依赖特征排序，然后扫描，为了减少特征排序，XGBoost引入一种名为block的数据存储结构，将数据存储在内存单元，进行排序。block中的数据以CSC格式存储。实际上源代码中XGBoost会把文件数据读入先生成CSR格式，然后转化为CSC格式。其中CSR格式如下：



举报





10

2

2

2

2

2

2

2

2

2

CSR包含非0数据块values，行偏移offsets，列下标indices。offsets数组大小为（总行数+1），CSR是对稠密矩阵的压缩，访问过程如下：

1. 根据行*i*得到偏移区间开始位置offsets[i]与区间结束位置offsets[i+1]-1，得到*i*行数据块values[offsets[i]..(offsets[i+1]-1)]，indices[offsets[i]..(offsets[i+1]-1)]，
2. 在列下标数据块中二分查找*j*，找不到则返回0，否则找到下标值*k*，返回values[offsets[i]+k]

从访问单个元素来说，从 $O(1)$ 时间复杂度升到 $O(\log N)$ ， N 为该行非稀疏数据项个数。但是如果遍历访问整行非0数据，则无需访问indices数组反而更低，因为少了大量的稀疏为0的数据访问。

CSC与CSR变量结构上并无差别，只是变量意义不同，其中values仍然为非0数据块，offsets为列偏移，即特征id对应数组，indices为行下标，XGBoost使用CSC主要用于对特征的全局预排序。预先将CSR数据转化为无序的CSC数据，遍历每个特征，并对每个特征*i*进行排序：sort(&value & values[offsets[i+1]-1])。全局特征排序后，后期节点分裂可以复用全局排序信息，而不需要重新排序。

Cache-aware Access

CSC存储优化会导致获取每个样本获取统计值而不连续，造成样本计算cache不断切换而导致cache-miss，XGBoost通过选择适当的block size来减小样本量带来的资源浪费以及大样本量带来的cache-miss之间的权衡问题，XGBoost选择的block size为 2^{16} 。

Out-of-core Computation

XGBoost中提出Out-of-core Computation优化，解决了在硬盘上读取数据耗时过长，吞吐量不足：

- 1) Block Compression基于block，数据分块，每块 2^{16} 个样例，使用16bit来存储offset。利用压缩算法将硬盘中的数据进行压缩，在读取数据过程中利用一个独立的线程对数据进行解压缩，将disk reading cost转换为解压缩所消耗的计算资源。
- 2) Block Sharding将数据shard到多块硬盘上，每块硬盘分配一个预取线程，将数据fetch到in-memory buffer中。训练线程交替读取多块buffer总体的吞吐量。

5. XGBoost算法复杂度

针对精确贪心算法，考虑数据样本量为 N ，特征数量为 M ，设置树的个数为 K ，树深为 D ，不考虑行采样与列采样，其时间复杂度分析如下：

1. 全局特征预排序，由于全局排序，后期节点再分裂可以复用全局排序信息，而不需要重新排序，因此排序复杂度为 $O(MN \log(N))$
2. 构建单树复杂度：由于XGBoost实现基于level-wise，每层的时间复杂度是为 $O(MN)$ ， K 颗树复杂度为 $O(KMND)$
3. 最终时间复杂度为： $O(MN \log(N)) + O(KMND)$ ，注意：跟论文的分析不同，主要按照笔者的理解，后期仔细分析后，如果有出入会

参考资料

1. Friedman Boosting框架论文：<https://statweb.stanford.edu/~jhf/ftp/trebst.pdf>
2. 陈天奇XGBoost论文：<https://arxiv.org/pdf/1603.02754.pdf>
3. XGBoost项目：<https://github.com/dmlc/xgboost>
4. GK Summary算法论文：<http://infolab.stanford.edu/~dadar/courses/cs361a/papers/quantiles.pdf>
5. A fast algorithm for approximate quantiles论文：<https://pdfs.semanticscholar.org/03a0/f978de91f70249dc39de758c49df4583.pdf>
6. wepon GDBT ppt：<http://202.38.196.91/cache/2/03/wepon.me/5aa84bcab4e621a09cc475c348590c35/gbdt.pdf>

8

8

8



cyber19

发布了9 原创文章 · 获赞 20 · 访问量 2万+

👍
10

私信

💬
2

想对作者说点什么



qq_40940180

1年前

你好 我想请问下 为什么obj化简为二次函数后 可以直接取最值 就保证最小化 难道二次项前面的系数Hj+λ必大于0吗? ?



manmantj

1年前

楼主，陈天奇论文说 ‘Weighted Quantile Sketch’ 可以应用于weighted data，请问何为weighted data？还有为什么 ‘Weighted Quantile Sketch’ 就可以应用data呢？

一文读懂机器学习大杀器XGBoost原理

阅读数 1681

【导读】XGBoost是boosting算法的其中一种。Boosting算法的思想是将许多弱分类器集成在一起形成一个强分类...

博文 | 来自: TensorFlowNews

XGBoost原理介绍-----个人理解版

阅读数 1万+

本人第一次写博客，这是篇算法总结的文章，希望能对大家的学习有所帮助。有什么错误之处，还望留言指出，希望...

博文 | 来自: oakley学习和总结...

XGBoost解析系列-数据加载

阅读数 3288

前言XGBoost数据加载1 DMatrixLoad主流程2 解析器parser构建过程3 DMatrix对象构建过程0.前言 本文主要...

博文 | 来自: cyber的博客

XGBoost原理详解

阅读数 1491

(作者：陈玒功) XGBoost是时下集成学习中最火的算法，效率非常之高，它的基础是集成学习中的GBDT，但是在...

博文 | 来自: 陈玒功的博客

亿速云高防服务器BGP智能专线

亿速云高防服务器，20+行业领袖视频推荐 前10大游戏公司CEO鼎力支持，速度快稳定有保障

广告 亿速云

通俗、有逻辑的写一篇说下Xgboost的原理，供讨论参考

阅读数 6万+

初看Xgboost，翻了多篇博客发现关于xgboost原理的描述实在难以忍受，缺乏逻辑性，写一篇供讨论。——以下是...

博文 | 来自: 就俩字，咱们要做...

xgboost 算法原理

阅读数 3万+

1、xgboost是什么全称：eXtremeGradientBoosting作者：陈天奇(华盛顿大学博士)基础：GBDT所属：boosting...

博文 | 来自: 哆啦A梦的博客

XGBoost的基本原理

阅读数 1万+

XGBoost原理与实践

博文 | 来自: Y学习使我快乐V的...

XGBoost解析系列--源码主流程

阅读数 8906

前言入口过程Train过程1Train主框架2UpdateOnelter流程21LazyInitDMatrix过程22PredictRaw过程23obj_-GetG...

博文 | 来自: cyber的博客

开源一个功能完整的SpringBoot项目框架

阅读数 7万+

福利来了，给大家带来一个福利。最近想了解一下有关Spring Boot的开源项目，看了很多开源的框架，大多是一些d...

博文

XGBoost解析系列-数据加载_cyber的博客-CSDN博客

XGBoost解析系列--源码主流程 - cyber的博客 - CSDN博客



举报



| | | | | |
|--|--|--|--|--|
| <div><div>CSDN</div><div>首页 博客 学院 下载 论坛 问答 活动 专题 招聘 APP VIP会员 续费8折</div><div>Python工程师</div><div>🔍</div></div> | | <div>🔥</div> | | |
| 前言 XGBoost (eXtreme Gradient Boosting) 全名叫极端梯度提升, XGBoost是集成学习方法的王牌, 在Kaggle... <div>博文 来自: htbeker的</div> | | <div>👍 10</div> <div>🔗</div> <div>💬 2</div> | | |
| XGBoost解析系列-准备 - cyber的博客 - CSDN博客 | | | | |
| XGBoost原理详解_机器学习_Panpan Wei的博客-CSDN博客 | | | | |
| 通俗易懂地给女朋友讲：线程池的内部原理 | | 阅读量 10万+ | | |
| 餐盘在灯光的照耀下格外晶莹洁白, 女朋友拿起红酒杯轻轻地抿了一小口, 对我说: “经常听你说线程池, 到底线程... <div>博文 来自: 万猫学社</div> | | <div>☆</div> <div>📱</div> <div><</div> <div>></div> <div>赏</div> | | |
| <div><div></div><div>磐创 AI 467篇文章 排名:3000+<div>关注</div></div></div> | | <div><div></div><div>Oakley_oupeng 1篇文章 排名:千里之外<div>关注</div></div></div> | | |
| <div><div></div><div>小白白白又 97篇文章 排名:千里之外<div>关注</div></div></div> | | | | |
| XGBoost原理解析_Dreamyx的博客-CSDN博客 | | | | |
| XGBoost原理介绍 - 糖葫芦君的博客 - CSDN博客 | | | | |
| 天天学JAVA-JAVA基础(6) | | 阅读量 2万+ | | |
| 如果觉得我写的还行, 请关注我的博客并且点个赞哟。本文主要介绍JAVA 中最常使用字符串常量String相关知识。1... <div>博文 来自: 穿越清华</div> | | | | |
| Xgboost算法推导及分析 | | 阅读量 6072 | | |
| Author: DivinerShi Xgboost其实就是gbdt的一个改进版本, 但是因为效果好, 工程建设完善, 所以经常和传统的g... <div>博文 来自: 快来学习鸭 ~ ~ ~</div> | | | | |
| Xgboost原理详解_机器学习_格物致知-CSDN博客 | | | | |
| XGBoost的基本原理 - Y学习使我快乐V的博客 - CSDN博客 | | | | |
| XGBoost——机器学习（理论+图解+安装方法+python代码） | | 阅读量 5万+ | | |
| 目录一、集成算法思想二、XGBoost基本思想三、MacOS安装XGBoost四、用python实现XGBoost算法在竞赛题中... <div>博文 来自: huacha_的博客</div> | | | | |
| Xgboost原理介绍,通俗易懂 - mxg1022的博客 - CSDN博客 | | | | |
| 我的 Input框 不可能这么可爱 | | 阅读量 11万+ | | |
| 作者: 陈大鱼头github: KRISACHAN<input /> 标签是我们日常开发中非常常见的替换元素了, 但是最近在刷 wh... <div>博文 来自: 鱼头的Web海洋</div> | | | | |
| XGBoost设计思路与数学推导 | | 阅读量 5979 | | |
| XGBoost是由陈天奇大神设计的一套基于gbdt的可并行计算的机器学习工具。在kaggle、天池等大数据竞赛有着广... <div>博文 来自: 啦啦啦种太阳</div> | | | | |
| XGBoost解析系列-准备 | | 阅读量 2411 | | |
| 前言代码准备编译准备debug编译配置VS Code可视化调试0.前言 研究生期间有幸和各路大腿参加过些机器学习... <div>博文 来自: cyber的博客</div> | | | | |
| xgboost原理分析以及实践 | | 阅读量 8622 | | |
| 摘要本文在写完GBDT的三篇文章后本来就想写的, 但一直没有时间, 终于刚好碰上需要, 有空来写这篇关于xgboos... <div>博文 来自: SCUT_Sam</div> | | | | |
| XGBoost原理解析 | | 阅读量 6962 | | |
| 本文对XGBoost的数学原理做了深入细致的解析, 对其中一些数学公式的推导进行了详细说明, 目的就是要让刚接触... <div>博文 来自: Dreamyx的博客</div> | | | | |
| 2019年10月中国编程语言排行榜 | | 阅读量 2万+ | | |
| 2019年10月2日, 我统计了某招聘网站, 获得有效程序员招聘数据9万条。针对招聘信息, 提取编程语言关键字, 并... <div>博文 来自: 毛毛虫</div> | | <div>🔔</div> <div>📢 2万+</div> <div>⬆️ 3万+</div> | | |
| XGBoost原理介绍 | | 1.Introduction在这篇文章中, 我将介绍XGBoost (eXtremeGradientBoosting) , 一种treeboosting的可扩展机... <div>博文 来自: 糖葫芦君的</div> | | |
| 经典算法（5）杨辉三角 | | 杨辉三角 是经典算法, 这篇博客对它的算法思想进行了讲解, 并有完整的代码实现。 ... <div>博文 来自: 扬帆向海的博客</div> | | |

TA的个人主页 >

原创 9 粉丝 27 获赞 20 评论 4 访问 2万+

等级: 博客 已 周排名: 22万+
积分: 390 总排名: 17万+
勋章: 

关注

私信








LOUIS VUITTON

最新文章

XGBoost解析系列-数据加载
XGBoost解析系列--源码主流程
Fast Algorithm for GK Summary算法
分布式GK Summary算法
GK Summay算法 (ϵ -approximate ϕ -quantile)

分类专栏

| | | |
|--|----------|----|
|  | xgboost | 7篇 |
|  | c++ | 5篇 |
|  | quantile | 3篇 |
|  | debug | 1篇 |
|  | gdb | 1篇 |

归档

| | |
|----------|----|
| 2017年12月 | 2篇 |
| 2017年11月 | 7篇 |

热门文章

XGBoost解析系列--源码主流程
阅读数 8902
XGBoost解析系列-原理
阅读数 5886
GDB配置 (打印STL容器、VS code配置、
远程调试debug)
阅读数 3436
XGBoost解析系列-数据加载
阅读数 3288
XGBoost解析系列-准备

10

2

2

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

赏

最新评论

XGBoost解析系列-原理

qq_40940180: 你好 我想请问下 为什么obj化简为二次函数后 可以直接取最大值 就保证最小化 难...

XGBoost解析系列--源码主流程

sinat_24850467: 我想请教一下, 使用xgboost里的pairwise的话, 样本怎么构造啊? 有数据吗? ...

XGBoost解析系列-原理

u014712516: 楼主, 陈天奇论文说 'Weighted Quantile Sketch' 可以应用于weighted data, ...

XGBoost解析系列-准备

Lglrdjj: 有没Windows环境下的配置方法, 搞了一下午, 也没搞通



QQ客服

kefu@csdn.net

客服论坛

400-660-0108

工作时间 8:30-22:00

[关于我们](#) | [招聘](#) | [广告服务](#) | [网站地图](#)

京ICP备19004658号 经营性网站备案信息

公安备案号 11010502030143

©1999-2020 北京创新乐知网络技术有限

公司 网络110报警服务

北京互联网违法和不良信息举报中心

中国互联网举报中心 家长监护 版权申诉



10



2



赏



举报

