GBDT 算法

BDA 小组
中国科学院计算技术研究所

2015 年 10 月

1 GBDT 基础

1.1 算法概述

GBDT(Gradient Boosting Decision Tree) 是一种迭代的回归决策树算法。该算法生成的模型由多棵决策回归树构成，所有树的预测值累加起来即为最终的预测值。GBDT 是一种泛化能力 (Generalization) 较强的算法，目前被广泛应用与搜索排序，并经常出现在机器学习领域的竞赛中。

GBDT 由两个重要的概念组成：回归决策树 (Regression Decision Trees) 和梯度迭代 (Gradient Boosting)。

1.2 回归决策树

算法实现的第一个版本中，暂不支持离散特征的输入，因此这里假设输入特征均为连续值。

回归决策树是 GBDT 模型的基本组成单位，用来做回归分析。回归决策树采用二分递归分隔的方法，在每个节点处，选取某个特征的某个阈值，将当前节点所包含的样本集分成两个子样本集，形成两个子节点，再对子节点采用同样的方法继续分割，直到不能分割为止。因此，每个回归决策树都是一棵结构简单的二叉树。

1.2.1 树节点的分裂

用回归树上每个节点的方差 (Variance) 来表示该节点的不纯度 (Impurity)，方差越大，不纯度越高：

$$I_v(N) = \frac{1}{|S|^2} \sum_{i \in S} \sum_{j \in S} \frac{1}{2}(x_i - x_j)^2 \tag{1}$$

其中，S 表示节点所包含的样本集。

为了提高算法效率，使用公式 (1) 的另一种表示形式：

$$I_v(N) = \frac{1}{|S|}\left(\sum_{i\in S} {x_i}^2 - \frac{1}{|S|}\sum_{i\in S} x_i{}^2\right) \tag{2}$$

根据节点及其左右孩子节点的方差，用下式可以得到每次分裂的方差缩减量 (Variance Reduction)，方差缩减量越大，说明这次分裂效果越好：

$$R_v(N) = I_v(N) - \left(\frac{|S_1|}{|S|}I_v\left(N_1\right) + \frac{|S_2|}{|S|}I_v\left(N_2\right)\right) \tag{3}$$

其中， N 表示父亲节点所包含的样本集， N_1 表示左孩子节点所包含的样本集， N_2 表示右孩子节点所包含的样本集。

这样，在每个节点处，先分别对所有特征进行排序，并依次取每维特征的特征值作为阈值进行分裂尝试，找到可以使得方差缩减量最大的特征值 f_{ij} ，即为该节点的分裂标准。

1.2.2 树节点的预测值

当节点不能再继续分裂的时候，我们使用该节点上样本集的均值作为该节点所对应的预测值：

$$P_i = \frac{1}{|S_i|}\sum_{i\in S_i} x_i \tag{4}$$

其中， P_i 表示第 i 个叶子节点的预测值， S_i 表示第 i 个叶子节点上的样本集。

1.3 梯度迭代

梯度迭代 (Grandient Boosting) 告诉我们如何将回归决策树进行组合。

初始时，所有样本的预测值置为统一的值，该值使得损失函数的取值最小。在每一轮迭代的过程中，每个样本点使用损失函数的负梯度作为目标值构建回归树，更新预测值为本轮回归树的输出值及之前回归树的输出值之和。

梯度迭代的执行过程请参考伪代码 (1) 。

1.4 缩减

缩减 (Shrinkage) 的思想认为，在模型的迭代过程中，每次走一小步逐渐逼近结果的效果，要比每次都一大步很快逼近结果的方式更容易避免过拟合，同时也更容易收敛。

Algorithm 1 Gradient tree boosting for multiple additive regression trees

1: $f_0 \leftarrow argmin_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
2: for $m = 1 \rightarrow M$ do
3: for $i = 1 \rightarrow N$ do
4: $r_{im} \leftarrow -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}$
5: end for
6: Fit a regression tree to the targets r_{im} giving terminal regions $R_{jm}, j = 1, 2, ..., J_m$.
7: for $j = 1 \rightarrow J_m$ do
8: $\gamma_{jm} \leftarrow argmin_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$
9: end for
10: $f_m(x) \leftarrow f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$
11: end for
12: Output $\hat{f}(x) \leftarrow f_M(x)$

也就是说，我们不能完全信任每一棵决策树，每一棵决策树只能学到真理的一小部分，累加的时候只累加一小部分，通过多棵决策树来弥补不足。

因此，对伪代码 (1) 第 (10) 行的公式进行修改，如公式 (5) 所示。其中，ρ_m 通常被称作学习率 (Learning Rate)。

$$f_m(x) \leftarrow f_{m-1}(x) + \rho_m \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm}) \tag{5}$$

2 GBDT 实现

目前 GBDT 算法的实现包括单机和半分布式两个版本。

2.1 单机版本

2.1.1 回归决策树

单机版本的回归决策树主要包括以下几个类：

- bda.local.ml.DTree：由用户指定回归决策树参数，通过 fit 方法读取训练数据并输出回归树模型。
- bda.local.ml.model.DTreeModel：回归决策树模型类。该类包括训练参数及生成的二叉树数据结构。
- bda.local.ml.model.Node：回归决策树的节点类。该类记录了在对应节点处分裂所使用的特征编号及其分裂阈值。

- bda.local.ml.model.Stat：模型训练的过程中，节点的状态类。该类记录了对应节点所包含的训练数据范围及训练数据的个数、加和以及平方和，通过这些信息可以有效地降低模型训练的时间复杂度。
- bda.local.para.DTreePara：回归决策树参数类。它是决策树模型训练的依据。

DTree 在训练模型的时候，会从根节点开始，通过遍历所有特征及其所有可能的分裂点的形式，为每一个节点寻找一个最佳的分裂特征及分裂阈值，并进行分裂。这个过程会不断进行下去，直到所有的节点都不满足分裂条件为止。

2.1.2 梯度迭代

单机版本的梯度迭代由以下几个类组成：

- bda.local.ml.GBoost：由用户指定回归决策树及梯度迭代的参数，通过 fit 方法读取训练数据并输出 GBDT 模型。
- bda.local.ml.model.GBoostModel：GBDT 模型类。该类包括训练参数及生成的 GBDT 模型 (多个 DTreeModel 的集合)。
- bda.local.ml.para.GBoostPara：GBDT 参数类。它是 GBDT 模型训练的依据。

2.2 半分布式版本

GBDT 算法的分布式实现主要瓶颈在于回归决策树的分布式实现。

根据理论部分的介绍，我们知道在构建一棵回归树的过程中需要遍历所有特征及其所有可能的分裂点。假设数据集的大小为 N，而特征个数为 M，那么在不计排序开销的前提下，回归树构建每一层的复杂度依然高达 $O(N * M)$，这样的复杂度在大规模数据下的时间开销难以让人接受。

为了减低算法的复杂度，使其可以满足大数据的需求，mllib 采用的方式是在训练回归树模型之前，对数据进行采样，并对这些数据的特征值进行分箱操作，将分箱的阈值作为对应特征的备选分裂点。由于分箱的个数远远小于数据规模，这样，就可以将回归树构建每一层的复杂度由 $O(N * M)$ 降为 $O(N' * M)$，其中，N' 表示分箱所使用的阈值个数。

但是，GBoost 在分布式的实现过程中并没有采用这一思想，而是在训练每一个回归树的模型过程时，采样训练样本，使用有限个数的样本训练单机版回归树模型作为单个弱分类器。这样，只对梯度迭代算法进行了分布式实现，从而形成了半分布式版本。

2.2.1 梯度迭代

分布式版本的梯度迭代由以下几个类组成：

- bda.spark.ml.GBoost：由用户指定回归决策树及梯度迭代的参数，通过 fit 方法读取训练数据并输出 GBDT 模型。
- bda.spark.ml.model.GBoostModel：GBDT 模型类。该类包括训练参数及生成的 GBDT 模型 (多个 DTreeModel 的集合)。
- bda.spark.ml.para.GBoostPara：GBDT 参数类。它是 GBDT 模型训练的依据。

3 GBDT 评测

3.1 单机版本

在单机环境下，与时下流行的 GBDT 多线程开源实现 xgboost 进行了对比，如表 (1) 所示。

- 测试环境：单机
  - CPU: 1.3 GHz Intel Core i5
  - Memory: 4 GB 1600 MHz DDR3
- 数据集：cadata
  - 类型：回归数据集
  - 来源：http://lib.stat.cmu.edu/datasets/houses.zip
  - 大小：6.4M
  - # of data：20,640
  - # of feature：8
- 固定参数
  - xgboost：gamma(0), max_delata_step(0), subsample(1), colsample_byree(1), lambda(1), alpha(0), nthread(10), learning_rate(0.01)
  - bda.local.ml.GBoost：learning_rate(0.01)

3.2 半分布式版本

TODO

表 1: 单机版本评测

algorithm	num iter	max depth	min child weight	total time	train RMSE	test RMSE
xgboost	100	15	10	4s	97536.96	102027.12
	300	15	10	9s	48573.12	58397.00
	600	15	10	26s	21310.80	47144.61
	600	15	20	24s	28383.57	47840.43
	500	20	20	26s	28251.16	46656.16
algorithm	num iter	max depth	min node size	ave iter time	train RMSE	test RMSE
GBoost	100	15	10	857ms	46993.31	60005.90
	300	15	10	692ms	20494.19	47727.48
	600	15	10	896ms	15688.38	47232.18
	1000	15	10	798ms	12343.36	47144.03