原 关于树的几个ensemble模型的比较（GBDT、xgBoost、lightGBM、RF）

2017年03月31日 17:22:23 阅读数：7116

决策树的Boosting方法比较

原始的Boosting是在算法开始的时候，为每一个样本赋上一个权重值，初始的时候，大家都是一样重要的。在每一步训练中得到的模型，会使得数据点的估计有对有错，我们就在每一步结束后，增加分错的点的权重，减少分对的点的权重，这样使得某些点如果老是被分错，那么就会被“严重关注”，也就被赋上一个很高的权重。然后等进行了N次迭代（由用户指定），将会得到N个简单的分类器（basic learner），然后我们将它们组合起来（比如说可以对它们进行加权、或者让它们进行投票等），得到一个最终的模型。

xgboost能自动利用cpu的多线程，而且适当改进了gradient boosting，加了剪枝，控制了模型的复杂程度

□ 传统GBDT以CART作为基分类器，xgboost还支持线性分类器，这个时候xgboost相当于带L1和L2正则化项的逻辑斯蒂回归（分类问题）或者线性回归（回归问题）。

□ 传统GBDT在优化时只用到一阶导数信息，xgboost则对代价函数进行了二阶泰勒展开，同时用到了一阶和二阶导数。顺便提一下，xgboost工具支持自定义代价函数，只要函数可一阶和二阶求导。

□ xgboost在代价函数里加入了正则项，用于控制模型的复杂度。正则项里包含了树的叶子节点个数、每个叶子节点上输出的score的L2模的平方和。从Bias-variance tradeoff角度来讲，正则项降低了模型的variance，使学习出来的模型更加简单，防止过拟合，这也是xgboost优于传统GBDT的一个特性。

□ Shrinkage（缩减），相当于学习速率（xgboost中的eta）。xgboost在进行完一次迭代后，会将叶子节点的权重乘上该系数，主要是为了削弱每棵树的影响，让后面有更大的学习空间。实际应用中，一般把eta设置得小一点，然后迭代次数设置得大一点。（传统GBDT的实现也有学习速率）

□ 列抽样（column subsampling）。xgboost借鉴了随机森林的做法，支持列抽样，不仅能降低过拟合，还能减少计算，这也是xgboost异于传统gbdt的一个特性。

□ 对缺失值的处理。对于特征的值有缺失的样本，xgboost可以自动学习出它的分裂方向。

□ xgboost工具支持并行。注意xgboost的并行不是tree粒度的并行，xgboost也是一次迭代完才能进行下一次迭代的（第t次迭代的代价函数里包含了前面t-1次迭代的预测值）。xgboost的并行是在特征粒度上的。我们知道，决策树的学习最耗时的一个步骤就是对特征的值进行排序（因为要确定最佳分割点），xgboost在训练之前，预先对数据进行了排序，然后保存为block结构，后面的迭代中重复地使用这个结构，大大减小计算量。这个block结构也使得并行成为了可能，在进行节点的分裂时，需要计算每个特征的增益，最终选增益最大的那个特征去做分裂，那么各个特征的增益计算就可以开多线程进行。

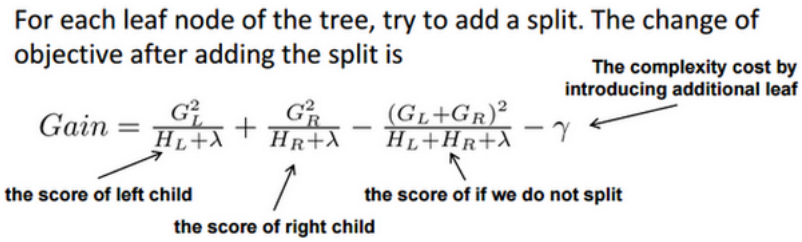
□ 可并行的近似直方图算法。树节点在进行分裂时，我们需要计算每个特征的每个分割点对应的增益，即用贪心法枚举所有可能的分割点。当数据无法一次载入内存或者在分布式情况下，贪心算法效率就会变得很低，所以xgboost还提出了一种可并行的近似直方图算法，用于高效地生成候选的分割点。xgboost的目标函数如下：

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Training loss　　Complexity of the Trees

其中正则项控制着模型的复杂度，包括了叶子节点数目T和leaf score的L2模的平方：

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



xgboost中[illegible]分裂时所采用的公式：

这个公式形[illegible]根ID3算法、CART算法是一致的，都是用分裂后的某种值减去分裂前的某种值，从而得到增益。为了限制树的生长，我[illegible]以加入阈值，当增益大于阈值时才让节点分裂，上式中的gamma即阈值，它是正则项里叶子节点数T的系数，所以xg[illegible]t在优化目标函数的同时相当于做了预剪枝。另外，上式中还有一个系数lambda，是正则项里leaf score的L2模平方的[illegible]，对leaf score做了平滑，也起到了防止过拟合的作用，这个是传统GBDT里不具备的特性。

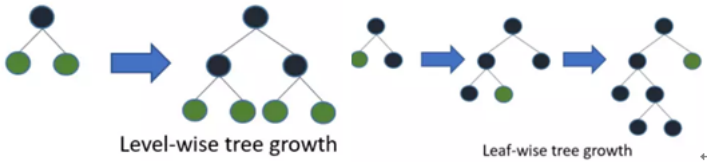
xgboost里[illegible]基学习器除了用tree(gbtree)，也可用线性分类器(gblinear)。而GBDT则特指梯度提升决策树算法。
xgboost相[illegible]普通gbm的实现，可能具有以下的一些优势：

1. 显式地将树模型的复杂度作为正则项加在优化目标
2. 公式推导里用到了二阶导数信息，而普通的GBDT只用到一阶
3. 允许使用column(feature) sampling来防止过拟合，借鉴了Random Forest的思想，sklearn里的gbm好像也有类似实现。

4.实现了一种分裂节点寻找的近似算法，用于加速和减小内存消耗。
5.节点分裂算法能自动利用特征的稀疏性。
6.data事先排好序并以block的形式存储，利于并行计算
7. penalty function Omega主要是对树的叶子数和叶子分数做惩罚，这点确保了树的简单性。。
8.支持分布式计算可以运行在MPI，YARN上，得益于底层支持容错的分布式通信框架rabit。

lightGBM：基于决策树算法的分布式梯度提升框架。

Lgb与xgBoost的区别：xgBoost使用的是pre-sorted算法（对所有特征都按照特征的数值进行预排序，在遍历分割点的时候用O(data)的代价找到一个特征上的最好分割点），能够更精确的找到数据分隔点；LightGBM使用的是histogram算法，占用的内存更低，数据分隔的复杂度更低。**决策树生长策略上**：XGBoost采用的是**level-wise生长策略**，能够同时分裂同一层的叶子，从而进行多线程优化，不容易过拟合；但不加区分的对待同一层的叶子，带来了很多没必要的开销（因为实际上很多叶子的分裂增益较低，没必要进行搜索和分裂）；LightGBM采用**leaf-wise生长策略**，每次从当前所有叶子中找到分裂增益最大（一般也是数据量最大）的一个叶子，然后分裂，如此循环；但会生长出比较深的决策树，产生过拟合（因此 LightGBM 在leaf-wise之上增加了一个最大深度的限制，在保证高效率的同时防止过拟合）。另一个比较巧妙的优化是 histogram 做差加速。一个容易观察到的现象：一个叶子的直方图可以由它的父亲节点的直方图与它兄弟的直方图做差得到。



GBDT（Gradient Boosting Decison Tree**）中的树都是回归树**，GBDT用来做回归预测，调整后也可以用于分类（设定阈值，大于阈值为正例，反之为负例），可以发现多种有区分性的特征以及特征组合。GBDT是把所有树的结论累加起来做最终结论的，**GBDT的核心就在于，每一棵树学的是之前所有树结论和的残差**，这个**残差就是一个加预测值后能得真实值的累加量**。比如A的真实年龄是18岁，但第一棵树的预测年龄是12岁，差了6岁，即残差为6岁。那么在第二棵树里我们把A的年龄设为6岁去学习，如果第二棵树真的能把A分到6岁的叶子节点，那累加两棵树的结论就是A的真实年龄；如果第二棵树的结论是5岁，则A仍然存在1岁的残差，第三棵树里A的年龄就变成1岁，继续学。 Boosting的最大好处在于，每一步的残差计算其实变相地增大了分错instance的权重，而已经分对的instance则都趋向于0。这样后面的树就能越来越专注那些前面被分错的instance。Gradient Boost与传统的Boost的区别是，每一次的计算是为了减少上一次的残差(residual)，而为了消除残差，我们可以在残差减少的梯度(Gradient)方向上建立一个新的模型。所以说，在Gradient Boost中，每个新的模型的建立是为了使得之前模型的残差往梯度方向减少。**Shrinkage（缩减）**的思想认为，每次走一小步逐渐逼近结果的效果，要比每次迈一大步很快逼近结果的方式更容易避免过拟合。即它不完全信任每一个棵残差树，它认为每棵树只学到了真理的一小部分，累加的时候只累加一小部分，通过多学几棵树弥补不足。**本质上，Shrinkage为每棵树设置了一个weight，累加时要乘以这个weight，但和Gradient并没有关系。**

优缺点： 它的非线性变换比较多，表达能力强，而且不需要做复杂的特征工程和特征变换。GBDT的缺点也很明显，Boost是一个串行过程，不好并行化，而且计算复杂度高，同时不太适合高维稀疏特征。

Adaboost是一种boost方法，它按分类对错，分配不同的weight，计算cost function时使用这些weight，从而让“错分的样本权重越大，使它们更被重视”

参考：http://blog.csdn.net/suranxu007/article/details/49910323

随机森林顾名思义，是用随机的方式建立一个森林，森林里面有很多的决策树组成，随机森林的每一棵决策树之间是没有关联的。在得到森林之后，当有一个新的输入样本进入的时候，就让森林中的每一棵决策树分别进行一下判断，看看这个样本应该属于哪一类（对于分类算法），然后看看哪一类被选择最多，就预测这个样本为那一类。有两个随机采样的过程，random forest对输入的数据要进行行、列的采样。对于行采样，采用有放回的方式，也就是在采样得到的样本集合中，可能有重复的样本。假设输入样本为N个，那么采样的样本也为N个。这样使得在训练的时候，每一棵树的输入样本都不是全部的样本，使得相对不容易出现over-fitting。然后进行列采样，从M个feature中，选择m个(m << M)。之后就是对采样之后的数据使用完全分裂的方式建立出决策树，这样决策树的某一个叶子节点要么是无法继续分裂的，要么里面的所有样本的都是指向的同一个分类。一般很多的决策树算法都一个重要的步骤 - 剪枝，但是这里不这样干，由于之前的两个随机采样的过程保证了随机性，所以就算不剪枝，也不会出现over-fitting。

随机森林优点：对于很多数据集表现良好，精确度较高；不易过拟合；可得到变量的重要性排序；既能处理离散型数据，也能处理连续型数据；不需要归一化；能够很好的处理缺失数据；易并行化。

用于分类：划分标准entropy，Gini 用于回归：Los:交叉熵

信息增益：熵（数据的不确定性程度）的减少；一个属性的信息增益量越大，这个属性作为一棵树的根节点就能使这棵树更简洁。信息增益=分裂前的熵 – 分裂后的熵H= - ∑pilog(pi) Info-Gain在面对类别较少的离散数据时效果较好，但如果面对连续的数据（如体重、身高、年龄、距离等），或者每列数据没有明显的类别之分（最极端的例子的该列所有数据都独一无二），即每个值对应一个样本

C4.5采用***信息增益率***：克服了ID3用信息增益选择属性时偏向选择取值多的属性的不足（某个属性存在大量的不同值，在划分时将每个值分为一个结点）。例：A起点10m/s，10s后20m/s，其差值大，B起点1m/s，1s后2m/s，但和A加速度相同。属性的重要性会随着其内在信息（Intrinsic Information）的增大而减小。它也有可能导致过分补偿，而选择那些内在信息很小的属性

$$Info = -\sum_{v \in value(A)} \frac{num(S_v)}{num(S)} log_2 \frac{num(S_v)}{num(S)},$$

这里Info为划分行为带来的信息，信息增益率如下计算：

$$Gain - ratio = \frac{IGain(S, A)}{Info}$$

基尼指数：$Gini(\mathrm{p}) = \sum_{k=1}^{K} p_k(1-\mathrm{p}_k) = 1 - \sum_{k=1}^{K} {p_k}^2$（假设有k个类）

Gini(p)表示p的不确定性，值越大表示不确定性越大。因此构造决策树时选择基尼指数最小的特征。

分类与回归树（CART）:二叉树形式，分类时：根据Gini指数选择划分特征；回归时：Los为平方损失函数，最小化均方误差选择划分特征，切分点（值）将数据切分成两部分，用平方误差最小的准则求解每个单元上的最优输出值（每个叶子节点上的预测值为所有样本的平均值）。

文章标签： 机器学习 算法 boost

个人分类： machine learning 决策树 boosting 随机森林 算法