逻辑回归 + GBDT模型融合实战!

原创 吴忠强 Datawhale 1周前

收录于话题
#数据项目专栏

12个

↑↑↑关注后"星标"Datawhale
每日干货 & 每月组队学习，不错过

Datawhale干货

作者：吴忠强，东北大学，Datawhale成员

一、GBDT+LR简介

协同过滤和矩阵分解存在的劣势就是仅利用了用户与物品相互行为信息进行推荐， 忽视了用户自身特征， 物品自身特征以及上下文信息等，导致生成的结果往往会比较片面。而这次介绍的这个模型是2014年由Facebook提出的GBDT+LR模型， 该模型利用GBDT自动进行特征筛选和组合， 进而生成新的离散特征向量， 再把该特征向量当做LR模型的输入， 来产生最后的预测结果， 该模型能够综合利用用户、物品和上下文等多种不同的特征， 生成较为全面的推荐结果，在CTR点击率预估场景下使用较为广泛。

下面首先会介绍逻辑回归和GBDT模型各自的原理及优缺点， 然后介绍GBDT+LR模型的工作原理和细节。

二、逻辑回归模型

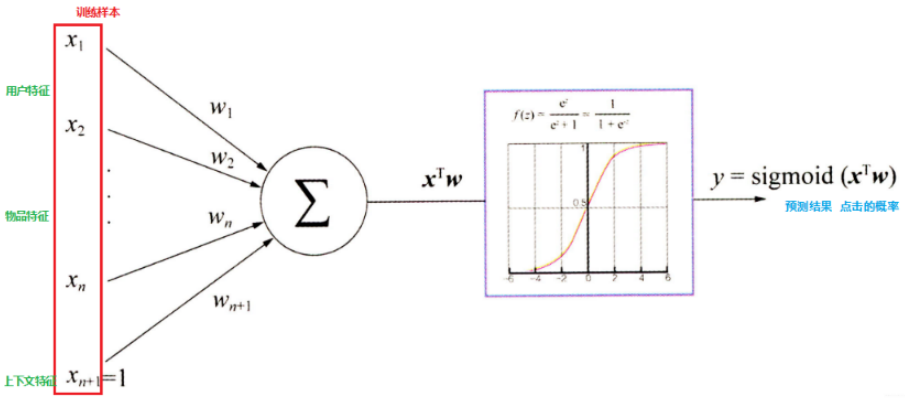
逻辑回归模型非常重要， 在推荐领域里面， 相比于传统的协同过滤， 逻辑回归模型能够综合利用用户、物品、上下文等多种不同的特征生成较为“全面”的推荐结果， 关于逻辑回归的更多细节， 可以参考下面给出的链接，这里只介绍比较重要的一些细节和在推荐中的应用。

逻辑回归是在线性回归的基础上加了一个 Sigmoid 函数（非线形）映射，使得逻辑回归成为了一个优秀的分类算法， 学习逻辑回归模型， 首先应该记住一句话：**逻辑回归假设数据服从伯努利分布,通过极大化似然函数的方法，运用梯度下降来求解参数，来达到将数据二分类的目的。**

相比于协同过滤和矩阵分解利用用户的物品“相似度”进行推荐， 逻辑回归模型将问题看成了一个分类问题， 通过预测正样本的概率对物品进行排序。这里的正样本可以是用户“点击”了某个商品或者“观看”了某个视频， 均是推荐系统希望用户产生的“正反馈”行为， 因此**逻辑回归模型将推荐问题转化成了一个点击率预估问题**。而点击率预测就是一个典型的二分类， 正好适合逻辑回归进行处理， 那么逻辑回归是如何做推荐的呢？过程如下：

1. 将用户年龄、性别、物品属性、物品描述、当前时间、当前地点等特征转成数值型向量
2. 确定逻辑回归的优化目标，比如把点击率预测转换成二分类问题， 这样就可以得到分类问题常用的损失作为目标， 训练模型
3. 在预测的时候， 将特征向量输入模型产生预测， 得到用户“点击”物品的概率
4. 利用点击概率对候选物品排序， 得到推荐列表

推断过程可以用下图来表示：



这里的关键就是每个特征的权重参数w， 我们一般是使用梯度下降的方式， 首先会先随机初始化参数w， 然后将特征向量（也就是我们上面数值化出来的特征）输入到模型， 就会通过计算得到模型的预测概率， 然后通过对目标函数求导得到每个w的梯度， 然后进行更新w

这里的目标函数长下面这样：

$$J(w)=-\frac{1}{m}\left(\sum_{i=1}^{m}\left(y^{i}\log f_{w}\left(x^{i}\right)+\left(1-y^{i}\right)\log\left(1-f_{w}\left(x^{i}\right)\right)\right)\right.$$

求导之后的方式长这样：

$$w_{j}\leftarrow w_{j}-\gamma\frac{1}{m}\sum_{i=1}^{m}\left(f_{w}\left(x^{i}\right)-y^{i}\right)x_{j}^{i}$$

这样通过若干次迭代， 就可以得到最终的w了， 关于这些公式的推导，可以参考下面给出的文章链接， 下面我们分析一下逻辑回归模型的优缺点。

优点：

1. LR模型形式简单，可解释性好，从特征的权重可以看到不同的特征对最后结果的影响。
2. 训练时便于并行化，在预测时只需要对特征进行线性加权，所以**性能比较好**，往往适合处理**海量id类特征**，用id类特征有一个很重要的好处，就是**防止信息损失**（相对于范化的 CTR 特征），对于头部资源会有更细致的描述
3. 资源占用小,尤其是内存。在实际的工程应用中只需要存储权重比较大的特征及特征对应的权重。
4. 方便输出结果调整。逻辑回归可以很方便的得到最后的分类结果，因为输出的是每个样本的概率分数，我们可以很容易的对这些概率分数进行cutoff，也就是划分阈值(大于某个阈值的是一类，小于某个阈值的是一类)

当然， 逻辑回归模型也有一定的局限性。

1. 表达能力不强， 无法进行特征交叉， 特征筛选等一系列“高级“操作（这些工作都得人工来干， 这样就需要一定的经验， 否则会走一些弯路）， 因此可能造成信息的损失
2. 准确率并不是很高。因为这毕竟是一个线性模型加了个sigmoid， 形式非常的简单(非常类似线性模型)，很难去拟合数据的真实分布
3. 处理非线性数据较麻烦。逻辑回归在不引入其他方法的情况下，只能处理线性可分的数据，如果想处理非线性， 首先对连续特征的处理需要先进行**离散化**（离散化的目的是为了引入非线性），如上文所说，人工分桶的方式会引入多种问题。
4. LR 需要进行**人工特征组合**，这就需要开发者有非常丰富的领域经验，才能不走弯路。这样的模型迁移起来比较困难，换一个领域又需要重新进行大量的特征工程。

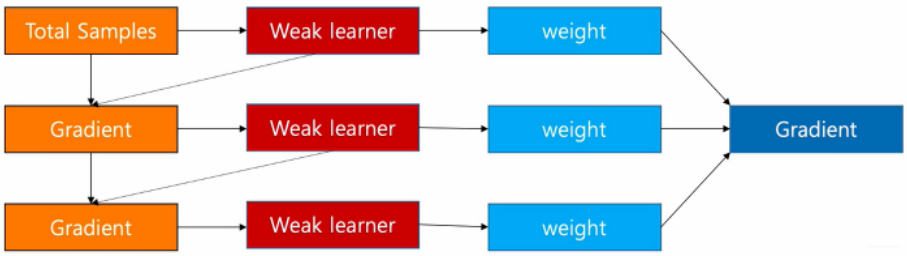
所以如何**自动发现有效的特征、特征组合，弥补人工经验不足，缩短LR特征实验周期**，是亟需解决的问题， 而GBDT模型， 正好可以**自动发现特征并进行有效组合**。注：在Datawhale公众

号后台回复【数据项目】可进项目专栏群，和作者等一起学习交流。

三、GBDT模型

GBDT全称梯度提升决策树，在传统机器学习算法里面是对真实分布拟合的最好的几种算法之一，在前几年深度学习还没有大行其道之前，gbdt在各种竞赛是大放异彩。原因大概有几个，一是效果确实挺不错。二是即可以用于分类也可以用于回归。三是可以筛选特征， 所以这个模型依然是一个非常重要的模型。

GBDT是通过采用加法模型(即基函数的线性组合），以及不断减小训练过程产生的误差来达到将数据分类或者回归的算法， 其训练过程如下：



gbdt通过多轮迭代， 每轮迭代会产生一个弱分类器， 每个分类器在上一轮分类器的残差基础上进行训练。gbdt对弱分类器的要求一般是足够简单， 并且低方差高偏差。因为训练的过程是通过降低偏差来不断提高最终分类器的精度。由于上述高偏差和简单的要求，每个分类回归树的深度不会很深。最终的总分类器是将每轮训练得到的弱分类器加权求和得到的（也就是加法模型）。

关于GBDT的详细细节，依然是可以参考下面给出的链接。这里想分析一下GBDT如何来进行二分类的，因为我们要明确一点就是**gbdt 每轮的训练是在上一轮的训练的残差基础之上进行训练的**， 而这里的残差指的就是当前模型的负梯度值， 这个就要求每轮迭代的时候，弱分类器的输出的结果相减是有意义的， 而**gbdt 无论用于分类还是回归一直都是使用的CART 回归树**， 那么既然是回归树， 是如何进行二分类问题的呢?

GBDT 来解决二分类问题和解决回归问题的本质是一样的，都是通过不断构建决策树的方式，使预测结果一步步的接近目标值， 但是二分类问题和回归问题的损失函数是不同的， 关于GBDT在回归问题上的树的生成过程， 损失函数和迭代原理可以参考给出的链接， 回归问题中一般使用的是平方损失， 而二分类问题中， GBDT和逻辑回归一样， 使用的下面这个：

$$L = \arg\min \left[\sum_{i}^{n} -(y_i \log(p_i) + (1 - y_i) \log(1 - p_i)) \right]$$

其中， y_i是第i个样本的观测值， 取值要么是0要么是1， 而p_i是第i个样本的预测值， 取值是0-1之间的概率，由于我们知道GBDT拟合的残差是当前模型的负梯度， 那么我们就需要求出这个模型的导数， 即$\frac{dL}{dp_i}$， 对于某个特定的样本， 求导的话就可以只考虑它本身， 去掉加和号， 那么就变成了$\frac{dl}{dp_i}$， 其中l如下：

$$\begin{aligned} l &= -y_i \log(p_i) - (1-y_i)\log(1-p_i) \\ &= -y_i \log(p_i) - \log(1-p_i) + y_i \log(1-p_i) \\ &= -y_i \left(\log\left(\frac{p_i}{1-p_i}\right)\right) - \log(1-p_i) \end{aligned}$$

如果对逻辑回归非常熟悉的话， $\left(\log\left(\frac{p_i}{1-p_i}\right)\right)$一定不会陌生吧， 这就是对几率比取了个对数， 并且在逻辑回归里面这个式子会等于θX， 所以才推出了$p_i = \frac{1}{1+e^{-\theta X}}$的那个形式。这里令$\eta_i = \frac{p_i}{1-p_i}$, 即$p_i = \frac{\eta_i}{1+\eta_i}$, 则上面这个式子变成了：

$$\begin{aligned} l &= -y_i \log(\eta_i) - \log\left(1 - \frac{e^{\log(\eta_i)}}{1+e^{\log(\eta_i)}}\right) \\ &= -y_i \log(\eta_i) - \log\left(\frac{1}{1+e^{\log(\eta_i)}}\right) \\ &= -y_i \log(\eta_i) + \log\left(1+e^{\log(\eta_i)}\right) \end{aligned}$$

这时候，我们对$log(\eta_i)$求导， 得

$$\frac{dl}{d\log(\eta_i)} = -y_i + \frac{e^{\log(\eta_i)}}{1+e^{\log(\eta_i)}} = -y_i + p_i$$

这样， 我们就得到了某个训练样本在当前模型的梯度值了， 那么残差就是$y_i - p_i$。GBDT二分类的这个思想，其实和逻辑回归的思想一样，**逻辑回归是用一个线性模型去拟合$P(y=1|x)$这个事件的对数几率$log\frac{p}{1-p} = \theta^T x$**， GBDT二分类也是如此， 用一系列的梯度提升树去拟合这个对数几率， 其分类模型可以表达为：

$$P(Y=1 \mid x) = \frac{1}{1+e^{-F_M(x)}}$$

下面我们具体来看GBDT的生成过程， 构建分类GBDT的步骤有两个：

1. 初始化GBDT

和回归问题一样， 分类 GBDT 的初始状态也只有一个叶子节点，该节点为所有样本的初始预测值，如下：

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y, \gamma)$$

上式里面， F代表GBDT模型， F_0是模型的初识状态， 该式子的意思是找到一个γ，使所有样本的 Loss 最小，在这里及下文中，γ都表示节点的输出，即叶子节点， 且它是一个 $log(\eta_i)$形式的值(回归值)，在初始状态，$\gamma = F_0$。

下面看例子(该例子来自下面的第二个链接)， 假设我们有下面3条样本：

喜欢爆米花	年龄	颜色偏好	喜欢看电影
Yes	12	Blue	Yes
No	87	Green	Yes
No	44	Blue	No

我们希望构建 GBDT 分类树，它能通过「喜欢爆米花」、「年龄」和「颜色偏好」这 3 个特征来预测某一个样本是否喜欢看电影。 我们把数据代入上面的公式中求Loss:

$$\text{Loss} = L(1, \gamma) + L(1, \gamma) + L(0, \gamma)$$

为了令其最小， 我们求导， 且让导数为0， 则：

$$\text{Loss} = p - 1 + p - 1 + p = 0$$

于是， 就得到了初始值$p = \frac{2}{3} = 0.67, \gamma = log(\frac{p}{1-p}) = 0.69$, 模型的初识状态$F_0(x) = 0.69$

2. 循环生成决策树

这里回忆一下回归树的生成步骤， 其实有4小步， 第一就是计算负梯度值得到残差， 第二步是用回归树拟合残差， 第三步是计算叶子节点的输出值， 第四步是更新模型。下面我们一一来看：

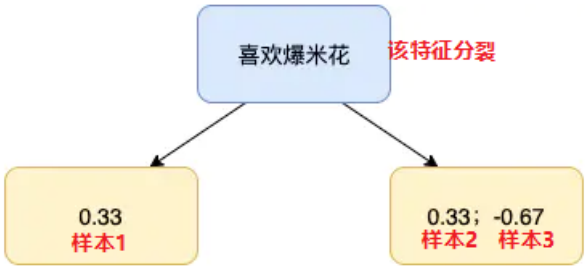
计算负梯度得到残差：

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$$

此处使用$m-1$棵树的模型， 计算每个样本的残差r_{im}, 就是上面的$y_i - pi$, 于是例子中， 每个样本的残差：

样本 i	喜欢看电影	第1棵树的残差 r_{i1}
1	Yes	1-0.67=0.33
2	Yes	1-0.67=0.33
3	No	0-0.67=-0.67

使用回归树来拟合r_{im}， 这里的i表示样本哈，回归树的建立过程可以参考下面的链接文章，简单的说就是遍历每个特征， 每个特征下遍历每个取值， 计算分裂后两组数据的平方损失， 找到最小的那个划分节点。假如我们产生的第2棵决策树如下：



对于每个叶子节点j, 计算最佳残差拟合值

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$$

意思是， 在刚构建的树m中， 找到每个节点j的输出γ_{jm}, 能使得该节点的loss最小。那么我们看一下这个γ的求解方式， 这里非常的巧妙。首先， 我们把损失函数写出来， 对于左边的第一个样本， 有

$$L(y_1, F_{m-1}(x_1) + \gamma) = -y_1(F_{m-1}(x_1) + \gamma) + \log\left(1 + e^{F_{m-1}(x_1)+\gamma}\right)$$

这个式子就是上面推导的l， 因为我们要用回归树做分类， 所以这里把分类的预测概率转换成了对数几率回归的形式， 即$log(\eta_i)$， 这个就是模型的回归输出值。而如果求这个损失的最小值，我们要求导， 解出令损失最小的γ。但是上面这个式子求导会很麻烦， 所以这里介绍了一个技巧就是**使用二阶泰勒公式来近似表示该式， 再求导**， 还记得伟大的泰勒吗?

$$f(x + \Delta x) \approx f(x) + \Delta x f'(x) + \frac{1}{2}\Delta x^2 f''(x) + O(\Delta x)$$

这里就相当于把$L(y_1, F_{m-1}(x_1))$当做常量$f(x)$， γ作为变量Δx， 将$f(x)$二阶展开：

$$L(y_1, F_{m-1}(x_1) + \gamma) \approx L(y_1, F_{m-1}(x_1)) + L'(y_1, F_{m-1}(x_1))\gamma + \frac{1}{2}L''(y_1, F_{m-1}(x$$

这时候再求导就简单了

$$\frac{dL}{d\gamma} = L'(y_1, F_{m-1}(x_1)) + L''(y_1, F_{m-1}(x_1))\gamma$$

Loss最小的时候， 上面的式子等于0， 就可以得到γ:

$$\gamma_{11} = \frac{-L'(y_1, F_{m-1}(x_1))}{L''(y_1, F_{m-1}(x_1))}$$

因为分子就是残差(上述已经求到了)， 分母可以通过对残差求导，得到原损失函数的二阶导:

$$\begin{aligned}
L''(y_1, F(x)) &= \frac{dL'}{d\log(\eta_1)} \\
&= \frac{d}{d\log(\eta_1)}\left[-y_i + \frac{e^{\log(\eta_1)}}{1+e^{\log(\eta_1)}}\right] \\
&= \frac{d}{d\log(\eta_1)}\left[e^{\log(\eta_1)}\left(1+e^{\log(\eta_1)}\right)^{-1}\right] \\
&= e^{\log(\eta_1)}\left(1+e^{\log(\eta_1)}\right)^{-1} - e^{2\log(\eta_1)}\left(1+e^{\log(\eta_1)}\right)^{-2} \\
&= \frac{e^{\log(\eta_1)}}{\left(1+e^{\log(\eta_1)}\right)^2} \\
&= \frac{\eta_1}{(1+\eta_1)}\frac{1}{(1+\eta_1)} \\
&= p_1(1-p_1)
\end{aligned}$$

这时候， 就可以算出该节点的输出:

$$\gamma_{11} = \frac{r_{11}}{p_{10}(1-p_{10})} = \frac{0.33}{0.67 \times 0.33} = 1.49$$

这里的下面γ_{jm}表示第m棵树的第j个叶子节点。接下来是右边节点的输出， 包含样本2和样本3， 同样使用二阶泰勒公式展开:

$$\begin{aligned}
&L(y_2, F_{m-1}(x_2) + \gamma) + L(y_3, F_{m-1}(x_3) + \gamma) \\
&\approx L(y_2, F_{m-1}(x_2)) + L'(y_2, F_{m-1}(x_2))\gamma + \tfrac{1}{2}L''(y_2, F_{m-1}(x_2))\gamma^2 \\
&+L(y_3, F_{m-1}(x_3)) + L'(y_3, F_{m-1}(x_3))\gamma + \tfrac{1}{2}L''(y_3, F_{m-1}(x_3))\gamma^2
\end{aligned}$$

求导， 令其结果为0，就会得到， 第1棵树的第2个叶子节点的输出:

$$\begin{aligned}
\gamma_{21} &= \frac{-L'(y_2, F_{m-1}(x_2)) - L'(y_3, F_{m-1}(x_3))}{L''(y_2, F_{m-1}(x_2)) + L''(y_3, F_{m-1}(x_3))} \\
&= \frac{r_{21} + r_{31}}{p_{20}(1-p_{20}) + p_{30}(1-p_{30})} \\
&= \frac{0.33 - 0.67}{0.67 \times 0.33 + 0.67 \times 0.33} \\
&= -0.77
\end{aligned}$$

可以看出， 对于任意叶子节点， 我们可以直接计算其输出值：

$$\gamma_{jm} = \frac{\sum_{i=1}^{R_{ij}} r_{im}}{\sum_{i=1}^{R_{ij}} p_{i,m-1}(1-p_{i,m-1})}$$

最后，更新模型$F_m(x)$：

$$F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_m$$

这样， 通过多次循环迭代， 就可以得到一个比较强的学习器$F_m(x)$。

下面分析一下GBDT的优缺点：

我们可以把树的生成过程理解成**自动进行多维度的特征组合**的过程，从根结点到叶子节点上的整个路径(多个特征值判断)，才能最终决定一棵树的预测值， 另外，对于**连续型特征**的处理，GBDT 可以拆分出一个临界阈值，比如大于 0.027 走左子树，小于等于 0.027（或者 default 值）走右子树，这样很好的规避了人工离散化的问题。这样就非常轻松的解决了逻辑回归那里**自动发现特征并进行有效组合**的问题， 这也是GBDT的优势所在。

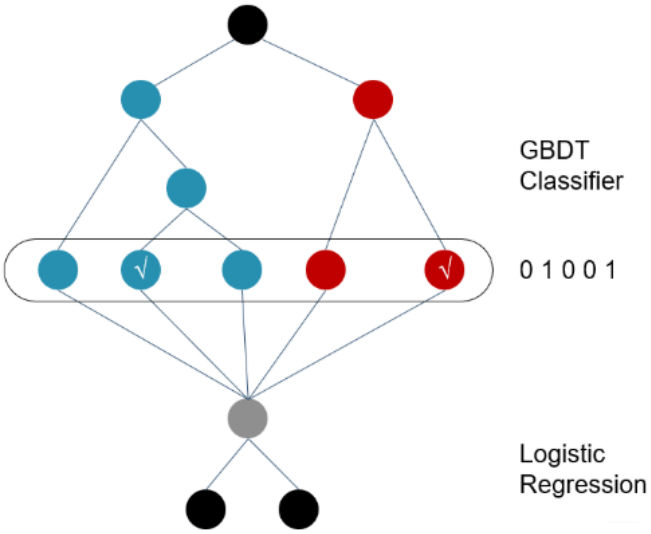
但是GBDT也会有一些局限性， 对于**海量的 id 类特征**，GBDT 由于树的深度和棵树限制（防止过拟合），不能有效的存储；另外海量特征在也会存在性能瓶颈，当 GBDT 的 one hot 特征大于 10 万维时，就必须做分布式的训练才能保证不爆内存。所以 GBDT 通常配合少量的反馈 CTR 特征来表达，这样虽然具有一定的范化能力，但是同时会有**信息损失**，对于头部资源不能有效的表达。

所以， 我们发现其实**GBDT和LR的优缺点可以进行互补**。

四、GBDT+LR模型

2014年， Facebook提出了一种利用GBDT自动进行特征筛选和组合， 进而生成新的离散特征向量， 再把该特征向量当做LR模型的输入， 来产生最后的预测结果， 这就是著名的GBDT+LR模型了。GBDT+LR 使用最广泛的场景是CTR点击率预估，即预测当给用户推送的广告会不会被用户点击。

有了上面的铺垫， 这个模型解释起来就比较容易了， 模型的总体结构长下面这样：



训练时，GBDT 建树的过程相当于自动进行的特征组合和离散化，然后从根结点到叶子节点的这条路径就可以看成是不同特征进行的特征组合，用叶子节点可以唯一的表示这条路径，并作为一个离散特征传入 LR 进行二次训练。

比如上图中， 有两棵树，x为一条输入样本，遍历两棵树后，x样本分别落到两颗树的叶子节点上，每个叶子节点对应LR一维特征，那么通过遍历树，就得到了该样本对应的所有LR特征。构造的新特征向量是取值0/1的。比如左树有三个叶子节点，右树有两个叶子节点，最终的特征即为五维的向量。对于输入x，假设他落在左树第二个节点，编码[0,1,0]，落在右树第二个节点则编码[0,1]，所以整体的编码为[0,1,0,0,1]，这类编码作为特征，输入到线性分类模型（LR or FM）中进行分类。

预测时，会先走 GBDT 的每棵树，得到某个叶子节点对应的一个离散特征(即一组特征组合)，然后把该特征以 one-hot 形式传入 LR 进行线性加权预测。

这个方案应该比较简单了， 下面有几个关键的点我们需要了解：

1. **通过GBDT进行特征组合之后得到的离散向量是和训练数据的原特征一块作为逻辑回归的输入， 而不仅仅全是这种离散特征**

2. 建树的时候用ensemble建树的原因就是一棵树的表达能力很弱，不足以表达多个有区分性的特征组合，多棵树的表达能力更强一些。GBDT每棵树都在学习前面棵树尚存的不足，迭代多少次就会生成多少棵树。

3. RF也是多棵树，但从效果上有实践证明不如GBDT。且GBDT前面的树，特征分裂主要体现对多数样本有区分度的特征；后面的树，主要体现的是经过前N颗树，残差仍然较大的少数

样本。优先选用在整体上有区分度的特征，再选用针对少数样本有区分度的特征，思路更加合理，这应该也是用GBDT的原因。

4. 在CRT预估中， GBDT一般会建立两类树(非ID特征建一类， ID类特征建一类)， AD，ID类特征在CTR预估中是非常重要的特征，直接将AD，ID作为feature进行建树不可行，故考虑为每个AD，ID建GBDT树。

- 非ID类树：不以细粒度的ID建树，此类树作为base，即便曝光少的广告、广告主，仍可以通过此类树得到有区分性的特征、特征组合
- ID类树：以细粒度 的ID建一类树，用于发现曝光充分的ID对应有区分性的特征、特征组合。

五、编程实践

下面我们通过kaggle上的一个ctr预测的比赛来看一下GBDT+LR模型部分的编程实践， 数据来源：https://github.com/zhongqiangwu960812/AI-RecommenderSystem/tree/master/GBDT%2BLR/data

我们回顾一下上面的模型架构， 首先是要训练GBDT模型， GBDT的实现一般可以使用xgboost， 或者lightgbm。训练完了GBDT模型之后， 我们需要预测出每个样本落在了哪棵树上的哪个节点上， 然后通过one-hot就会得到一些新的离散特征， 这和原来的特征进行合并组成新的数据集， 然后作为逻辑回归的输入，最后通过逻辑回归模型得到结果。

根据上面的步骤， 我们看看代码如何实现：

假设我们已经有了处理好的数据x_train, y_train。

1. 训练GBDT模型

GBDT模型的搭建我们可以通过XGBOOST， lightgbm等进行构建。比如：

```
gbm = lgb.LGBMRegressor(objective='binary',
                        subsample= 0.8,
                        min_child_weight= 0.5,
                        colsample_bytree= 0.7,
```

```
                        num_leaves=100,
                        max_depth = 12,
                        learning_rate=0.05,
                        n_estimators=10,
                        )

gbm.fit(x_train, y_train,
        eval_set = [(x_train, y_train), (x_val, y_val)],
        eval_names = ['train', 'val'],
        eval_metric = 'binary_logloss',
        # early_stopping_rounds = 100,
        )
```

2. 特征转换并构建新的数据集

通过上面我们建立好了一个gbdt模型， 我们接下来要用它来预测出样本会落在每棵树的哪个叶子节点上， 为后面的离散特征构建做准备， 由于不是用gbdt预测结果而是预测训练数据在每棵树上的具体位置， 就需要用到下面的语句：

```
model = gbm.booster_            # 获取到建立的树

# 每个样本落在每个树的位置 ， 下面两个是矩阵 (样本个数, 树的棵树) ， 每一个数字代表某个样本落在了某
gbdt_feats_train = model.predict(train, pred_leaf = True)
gbdt_feats_test = model.predict(test, pred_leaf = True)

# 把上面的矩阵转成新的样本-特征的形式， 与原有的数据集合并
gbdt_feats_name = ['gbdt_leaf_' + str(i) for i in range(gbdt_feats_train.shape[1])]
df_train_gbdt_feats = pd.DataFrame(gbdt_feats_train, columns = gbdt_feats_name)
df_test_gbdt_feats = pd.DataFrame(gbdt_feats_test, columns = gbdt_feats_name)

# 构造新数据集
train = pd.concat([train, df_train_gbdt_feats], axis = 1)
```

```
test = pd.concat([test, df_test_gbdt_feats], axis = 1)

train_len = train.shape[0]

data = pd.concat([train, test])
```

3. 离散特征的独热编码，并划分数据集

```
# 新数据的新特征进行读入编码
for col in gbdt_feats_name:
    onehot_feats = pd.get_dummies(data[col], prefix = col)
    data.drop([col], axis = 1, inplace = True)
    data = pd.concat([data, onehot_feats], axis = 1)


 # 划分数据集
train = data[: train_len]
test = data[train_len:]


x_train, x_val, y_train, y_val = train_test_split(train, target, test_size = 0.3, random_s
```

4. 训练逻辑回归模型作最后的预测

```
# 训练逻辑回归模型
lr = LogisticRegression()
lr.fit(x_train, y_train)
tr_logloss = log_loss(y_train, lr.predict_proba(x_train)[:, 1])
print('tr-logloss: ', tr_logloss)
val_logloss = log_loss(y_val, lr.predict_proba(x_val)[:, 1])
print('val-logloss: ', val_logloss)


# 预测
y_pred = lr.predict_proba(test)[:, 1]
```

上面我们就完成了GBDT+LR模型的基本训练步骤， 具体详细的代码可以参考链接。

六、课后思考

1. 为什么使用集成的决策树？为什么使用GBDT构建决策树而不是随机森林？
2. 面对高维稀疏类特征的时候(比如ID类特征)， 逻辑回归一般要比GBDT这种非线性模型好，为什么？

参考资料

- 王喆 - 《深度学习推荐系统》
- 决策树之 GBDT 算法 - 分类部分
- 深入理解GBDT二分类算法
- 逻辑回归、优化算法和正则化的幕后细节补充
- 梯度提升树GBDT的理论学习与细节补充
- 推荐系统遇上深度学习(十)--GBDT+LR融合方案实战
- CTR预估中GBDT与LR融合方案
- GBDT+LR算法解析及Python实现
- 常见计算广告点击率预估算法总结
- GBDT--分类篇

论文

- http://quinonero.net/Publications/predicting-clicks-facebook.pdf
- Predicting Clicks: Estimating the Click-Through Rate for New Ads\
- Greedy Fun tion Approximation : A Gradient Boosting