XGBoost

**Boosted Trees 介绍**

XGBoost 是 “Extreme Gradient Boosting”的简称，其中“Gradient Boosting”来源于附录1.Friedman的这篇论文。本文基于 gradient boosted tree ，中文可以叫梯度提升决策树，下面简称GBDT，同时也有简称GBRT，GBM。针对gradient boosted tree的细节也可以参考附录2.这篇网页，感兴趣的朋友可以前往[BigQuant](https://bigquant.com/?utm_source=jiqizhixin&utm_medium=jiqizhixin_article&utm_campaign=190124_L5)进一步学习实践。

**监督学习**

XGBoost 主要是用来解决有监督学习问题，此类问题利用包含多个特征的训练数据 xixi，来预测目标变量 yiyi。在我们深入探讨GBDT前，我们先来简单回顾一下监督学习的一些基本概念。

**模型与参数**

在监督学习中模型（model）表示一种数学函数，通过给定 xi 来对 yi 进行预测。以最常见的线性模型（linear model）举例来说，模型可以表述为 yi^=∑jθjxij，这是一个输入特性进行线性加权的函数。那么针对预测值的不同，可以分为回归或者分类两种。  
在监督学习中参数（parameters）是待定的部分，我们需要从数据中进行学习得到。在线性回归问题中，参数用θ来表示。

**目标函数：训练误差 + 正则化**

根据对 yi的不同理解，我们可以把问题分为，回归、分类、排序等。我们需要针对训练数据，尝试找到最好的参数。为此，我们需要定义所谓的目标函数，此函数用来度量参数的效果。  
这里需要强调的是，目标函数必须包含两个部分：训练误差和正则化。

https://image.jiqizhixin.com/uploads/editor/755ebc57-cc98-4dc4-94a5-78fb6e77ff4c/1548300497477.png

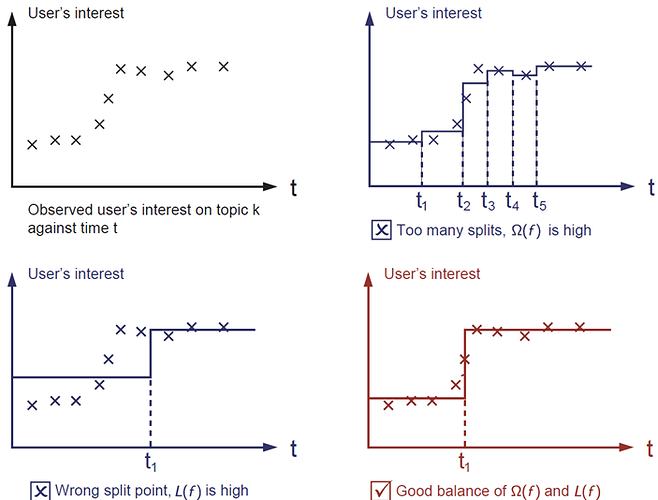
其中，L表示训练误差函数，Ω表示正则项。训练误差用来衡量模型在训练数据上的预测能力。比较典型的有用均方差来衡量。

https://image.jiqizhixin.com/uploads/editor/233a1397-d520-47de-ad56-9970415cec57/1548300516315.png

另外针对逻辑回归，比较常见的损失函数为Logistic函数：

https://image.jiqizhixin.com/uploads/editor/2cb2217d-0824-48c6-88fe-93777b1b6fc1/1548300532936.png

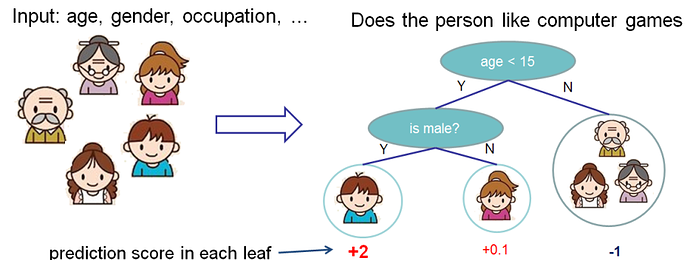
另外一个比较重要的部分就是正则项，这也是很多人容易忘记的部分。正则项是用来控制模型的复杂度，以防止过拟合（overfitting）。这听起来有点抽象，那么我们用下面的例子来说明。针对下面左上角的这幅图，我们需要找到一个阶梯函数来拟合图中的数据点集合。那么问题来了，下面剩下的三幅图中，哪一个你认为是最好的呢？

[](https://shequcdn.bigquant.com/community/uploads/default/original/1X/3376d7439481302a9e8f3e78a5d8806c45a8ecaa.png)

答案是用红色标注出来的这幅图。为什么呢？因为我们对于好的模型的判断依据是 简单（*simple*）并且 准确（*predictive*）。但这两者又是相互矛盾的，在机器学习中我们也把这两者也用 bias-variance 来表述。

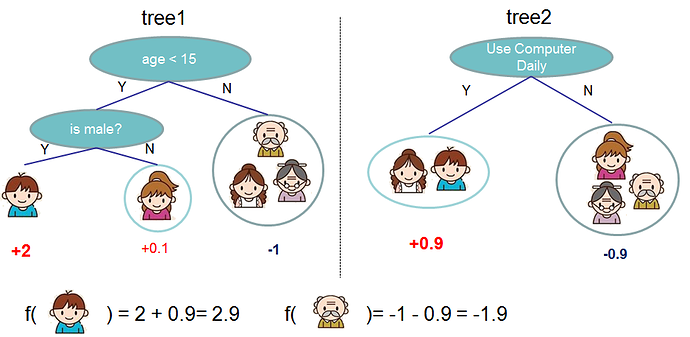
**复合树模型（Tree Ensemble）**

在前面我们已经介绍了监督学习，现在让我们开始了解树模型。首先先来了解一下xgboost所对应的模型：复合树模型。复合树模型是一组分类和回归树（classification and regression trees - CART）。这里我们举CART中的一个例子，一类分类器用来辨别某人是否喜欢计算机游戏。

[](https://shequcdn.bigquant.com/community/uploads/default/original/1X/fac975dcb4b25ddc2e1a5674ab5b3f119896bd99.png)

我们把家庭中的成员分到了不同的叶子节点，同时每个叶子节点上都有一个分数。CART与决策树相比，细微差别在于CART的叶子节点仅包含判断分数。在CART中，相比较于分类结果，每个叶子节点的分数给我们以更多的解释。这让CART统一优化节点更为容易，这在后面会有具体介绍。

通常情况下，在实践中往往一棵树是不够用的。这个时候往往需要把多棵树的预测结果综合起来，这就是所谓的复合树模型。

[](https://shequcdn.bigquant.com/community/uploads/default/original/1X/16a2487e639d02f00a99d488870a635f7a14a619.png)

上面就是由两棵树组成的复合树的例子。每棵树上的分数简单相加就得到了最终的分数。用数学式子可以表达如下：

https://image.jiqizhixin.com/uploads/editor/6c23ff43-860e-4064-9df6-896ba528674f/1548300578317.png

KK表示树的数目，ff是函数空间FF中的一个函数，FF表示CART的所有可能集合。所以我们的优化目标可以写作：

https://image.jiqizhixin.com/uploads/editor/01738ade-e526-447c-987c-c431f49a1494/1548300593551.png

现在问题来了，随机森林对应的模型是什么呢？对了，也是复合树模型。所以在模型的表述上，随机森林和提升树是一样的，他们俩的区别只是在于如何训练。这也就意味着，如果要写一个关于复合树模型的预测服务，我们只需要写一个就可以同时支持随机森林和提升树。

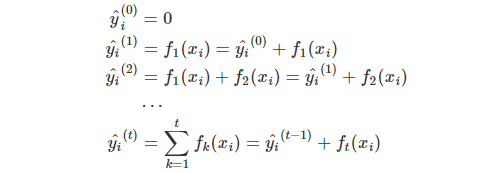
**提升树 （Tree Boosting）**

介绍了模型之后，让我们看看训练部分。那么我们是怎么训练这些树的呢？对于所有的监督学习模型，答案也都是同样，只需要做两件事，定义目标函数，然后优化它。  
假设我们有如下的目标函数（需要切记目标函数必须包含损失函数及正则项）

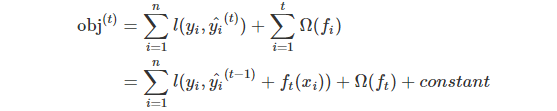
https://image.jiqizhixin.com/uploads/editor/61367e9b-d6d4-44e6-a81b-c06fb23d853d/1548300612676.png

**增量训练 （Additive Training）**

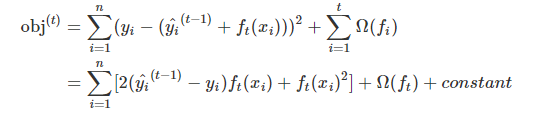
首先我们需要问的是，这些树的参数是什么？我们会发现，我们所要学习的就是这些 fifi方法，每个方法中定义树的结构以及叶子节点的分数。这比传统最优化问题要更难，传统最优化问题我们可以通过梯度来解决。而且我们无法在一次训练所有的树。相反，我们用增量（additive）的方式：每一步我们都是在前一步的基础上增加一棵树，而新增的这棵树是为修复上一颗树的不足。，我们把每tt步的预测用yi^(t)yi^(t)表示，这样我们就有了：



这里还有疑问的是，在每一步中如何确定哪棵树是我们需要的呢？一个很自然的想法就是，增加这棵树有助于我们的目标函数。



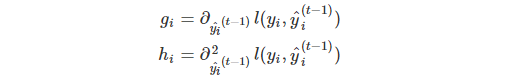
我们用MSE（均方差）作为损失函数，这样式子就变成了：



对于用MSE求出来的损失函数式子比较友好，包含一个一阶项和一个二次项。但是对于其他形式，就很难推导出这么友好的损失函数式子了。那么针对这种情形，我们就用泰勒展开公式（参考附录4，x取值yi^(t−1)+ft(xi)，a取值yi^(t−1)）来逼近：

https://image.jiqizhixin.com/uploads/editor/0026d0cd-42e7-4107-be69-84cd77201acb/1548300737293.png

其中 gi 和 hi 定义如下：



然后针对上述式子，我们删除常数项，那么在 tt 目标函数就变成：

https://image.jiqizhixin.com/uploads/editor/ed2a6b00-8f49-4a30-b559-5e4d78c451eb/1548300769597.png

选择新的一棵树，上述式子就是优化目标。这样的优化目标有一个优点，式子只需要考虑gi和hi。这就是xgboost为什么能支持自定义损失函数的原因。我们能够优化每一个损失函数，包括逻辑回归和加权逻辑回归，只需要把对应的gi和hi作为输入传入即可。

**模型复杂度**

现在讲讲正则化。那么如何定义 Ω(f)呢，在此之前，我们需要定义 f(x)：

https://image.jiqizhixin.com/uploads/editor/61c892ad-92f4-4036-9312-ec36195f9165/1548300815774.png

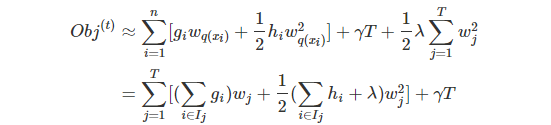
这里ww表示叶子节点上的分数所组成的向量，qq表示每个数据映射到相应叶子节点的对应关系函数，TT表示叶子节点的数量。在XGBoost中，我们用如下公式定义复杂度：

https://image.jiqizhixin.com/uploads/editor/1f10c403-48bd-4d04-b910-229684397007/1548300828413.png

当然还有其他公式来定义复杂度，但是我们发现上述式子在实践过程中表现很好。其他树相关的算法包不怎么认真对待正则化，甚至直接忽视掉。

**如何计算树叶子节点上的分数**

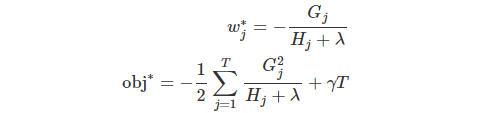
那么在增量学习过程中，如何选择这棵新增的树呢？要解决这个问题，我们先解决一下其中这个子问题：假设这棵树的结构已经确定了，如何来计算叶子节点上的分数？  
这一部分是推广过程中比较神奇的一个步骤。根据上述过程，我们写出第tt步树的目标值：



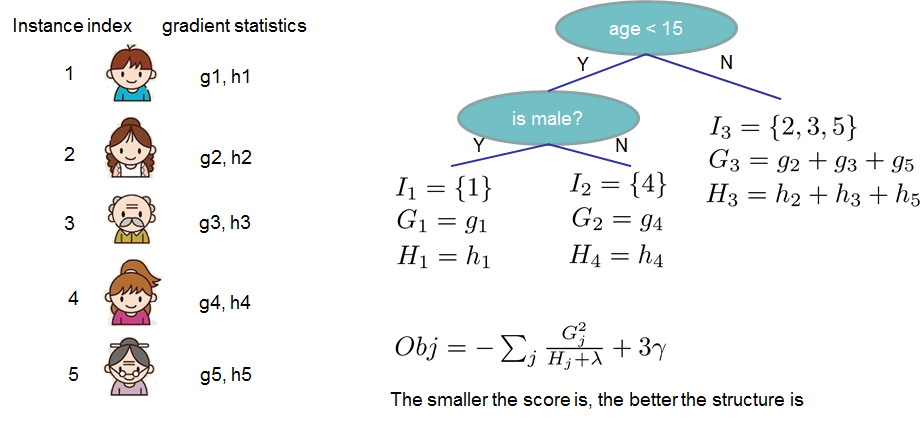
这里 Ij=i|q(xi)=j表示每个映射到第j个叶子节点对应的数据样本。需要注意的是，因为映射到相同叶子节点上的数据样本他们的分数是相同的，所以在第二行我们改变了一下求和∑顺序。同时我们令 Gj=∑i∈Ijgi以及 Hj=∑i∈Ijhi，那么上述公式简化为：

https://image.jiqizhixin.com/uploads/editor/83242f59-3c8e-4691-aa68-8f33bbc22263/1548300938563.png

在上述式子中，每一个wj是相互独立的，那么针对一元二次方程 Gjwj+1/2(Hj+λ)w2j而言，可以比较容易求出当新增的这棵树的结构 q(x) 已知的情况下，目标函数最小值下的 wj：



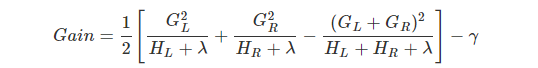
最后的式子计算的是树 q(x) 的优劣：

[](https://raw.githubusercontent.com/dmlc/web-data/master/xgboost/model/struct_score.png)

如果上面的式子看着比较复杂的话，那么根据上面的这幅图来看如何计算这些分数，就会显得更直观些。一旦树的结构已知的话，我们只需要通过计算每个节点上的 gigi 和 hihi ，然后把各个叶子节点上的这些数值加起来，用上述方程式就可以计算这棵树的优劣了。

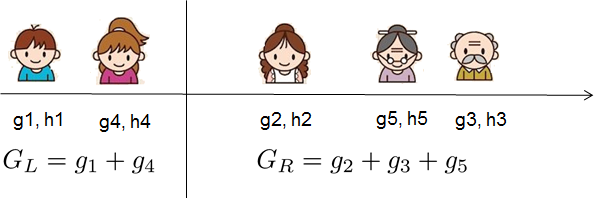
**如何学习树的结构**

现在我们已经知道一旦树的结构固定下来以后，如何来计算叶子节点上的分数，以及计算这棵树的优劣。那么关于现在我们要来解决如何来学习这棵树的结构。比较简单粗暴的方法就是遍历所有可能的树结构，然后从中找到最好的那棵树。但是这也是不切实际的，因为需要遍历的情况实在是太多了。所以我们来寻求一种贪婪的解法，就是在树的每个层构建的过程中，来优化目标。那么这里假设在某一层的构建过程中，假设特征已经选定，我们先如何进行二叉划分呢，以及是不是需要进行划分？我们可以通过下面的式子来计算划分之后，在目标上所获得的收益（这个收益越越好，负数表示收益为负）：



上面的这个式子可以分解为 1) 若是划分，划分后左边节点的收益 2) 或是划分，划分后右边节点的收益 3) 如不划分，原先节点的收益 4) 划分后正则项的收益。通过上述式子比较容易看到，当划分后叶子节点所带来的新增收益小于 γγ ，我们最好还是不要进行二叉划分，保留原样是最好的。这也是日后做剪枝的依据。

那么针对排序后的特征，我们所要做的就是遍历各种划分，找到一个最好的划分点，如下图表示。

[](https://raw.githubusercontent.com/dmlc/web-data/master/xgboost/model/split_find.png)

那么这里还有一个问题就是在构建树的结构过程中，在某一层如何进行特征选择呢？这里提供了一种比较简单的方式就是遍历每一种特征，然后根据上述式子的Gain，找到最大的Gain值对应的特征。

**关于XGBoost的最后几句话**

我们花了很长时间来讲解 Boosted Tree，那么XGBoost相较于Boosted Tree，又做了哪些额外的事情呢？XGBoost是遵循上述Boosted Tree思想的工程实现，但同时又考虑兼顾系统优化和机器学习原理，最大化的保证可扩展性、便捷性以及准确性。