[翻译] 提升树算法的介绍（Introduction to Boosted Trees）

1. 有监督学习的要素

XGBoost 适用于**有监督学习**问题。在此类问题中，我们使用多特征的训练数据集 x_i 去预测一个目标变量 y_i 。在专门学习树模型前，我们先回顾一下有监督学习的基本要素。

Elements of Supervised Learning

XGBoost is used for supervised learning problems, where we use the training data (with multiple features) x_i to predict a target variable y_i. Before we learn about trees specifically, let us start by reviewing the basic elements in supervised learning.

1.1 模型和参数

有监督学习的**模型**通常指这样的数学结构：预测值 y_i 是由给定输入值 x_i 决定的。一个常见的例子是线性模型，其中预测值 $\hat{y}_i = \sum_j \theta_j x_{ij}$ 是一个对输入特征值的加权线性组合。这个预测值可以有不同的解释，这取决于任务是回归还是分类。例如，在逻辑回归中，可通过预测值的逻辑变换获得样本被归为正面类别的概率；当我们需要将输出进行排序时，预测值也可作为一个排序得分值。

参数是我们需要从数据中学习得到的非确定部分。在线性回归问题中，参数就是系数 θ。我们通常用 θ 代表参数（事实上模型中有许多参数，这里只是粗浅的定义一下）。

Model and Parameters

The model in supervised learning usually refers to the mathematical structure of by which the prediction y_i is made from the input x_i. A common example is a linear model, where the prediction is given as $\hat{y}_i = \sum_j \theta_j x_{ij}$, a linear combination of weighted input features. The prediction value can have different interpretations, depending on the task, i.e., regression or classification. For example, it can be logistic transformed to get the probability of positive class in logistic regression, and it can also be used as a ranking score when we want to rank the outputs.
The parameters are the undetermined part that we need to learn from data. In linear regression problems, the parameters are the coefficients θ. Usually we will use θ to denote the parameters (there are many parameters in a model, our definition here is sloppy).

1.2 目标函数：训练损失+正则项

通过谨慎地选则 y_i，我们可以表达各种各样的任务，比如回归、分类、排序。**训练**模型意味着要寻求最佳参数 θ 使得能够最好地拟合训练数据 x_i 和标签 y_i。为了训练模型，我们需要定义**目标函数**来衡量这个模型拟合数据的效果有多好。

目标函数的一个显著特征是，它由**训练损失**和**正则项**两个部分组成：

$$obj(\theta) = L(\theta) + \Omega(\theta)$$

公告

昵称：thousfeet
园龄：3年5个月
粉丝：126
关注：25
+加关注

‹ 2019年7月 ›

日	一	二	三	四	五	六
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10

随笔分类(54)
2017级面向对象程序设计助教(9)
2017软工实践(27)
Hadoop(8)
Machine Learning(5)
编程之美(1)
算法&数构(4)

随笔档案(88)
2019年4月 (1)
2018年10月 (6)
2018年9月 (2)
2018年7月 (2)
2018年6月 (3)
2018年5月 (1)
2018年4月 (2)
2018年3月 (9)
2018年2月 (2)
2017年12月 (9)
2017年11月 (24)
2017年10月 (7)
2017年9月 (3)
2017年8月 (2)
2017年7月 (1)
2016年12月 (1)
2016年10月 (1)
2016年6月 (3)
2016年5月 (4)
2016年4月 (1)
2016年2月 (1)
2016年1月 (3)

关注我
0 0

友链
deepcs邵昊

积分与排名

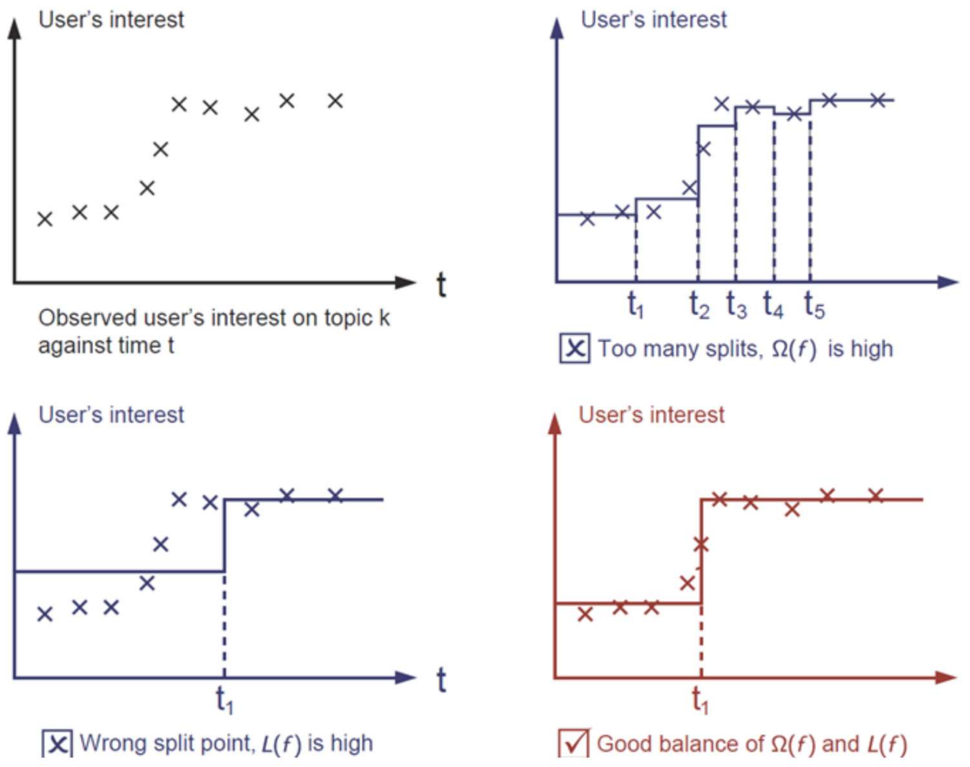
式中 L 是训练损失函数，Ω 是正则项。训练损失函数衡量的是模型在该训练数据集上的**预测能力**。L 通常选择**均方误差**，公式如下：

$$L(\theta) = \sum_i (y_i - \hat{y}_i)^2$$

另一个常用的损失函数是逻辑损失（logistic loss），被用于逻辑回归问题：

$$L(\theta) = \sum_i [y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) ln(1 + e^{\hat{y}_i})]$$

人们通常会忘记加正则项。实际上，**正则项**能够控制模型的复杂度，这有助于避免过拟合问题。这听起来有点抽象。让我们考虑下图中的这个问题：对于左上角图片中给出的输入数据，要求直观地给出一个阶梯函数的拟合结果，三个图中哪个方案的拟合效果最好？



正确答案已用红色标出了。你是否能够直观地感觉出这是一个合理的拟合结果？一般性的原则就是，我们希望得到一个**既简单又具备预测能力**的模型。在机器学习领域，这两者之间的折衷也被称为**偏差-方差权衡**（bias-variance tradeoff）。

Objective Function: Training Loss + Regularization

With judicious choices for yi, we may express a variety of tasks, such as regression, classification, and ranking. The task of training the model amounts to finding the best parameters θ that best fit the training data xi and labels yi. In order to train the model, we need to define the objective function to measure how well the model fit the training data.
A salient characteristic of objective functions is that they consist two parts: training loss and regularization term:

$$obj(\theta) = L(\theta) + \Omega(\theta)$$

where Lis the training loss function, and Ω is the regularization term. The training loss measures how predictive our model is with respect to the training data. A common choice of L is the mean squared error, which is given by

$$L(\theta) = \sum_i (y_i - \hat{y}_i)^2$$

Another commonly used loss function is logistic loss, to be used for logistic regression:

$$L(\theta) = \sum_i [y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) ln(1 + e^{\hat{y}_i})]$$

The regularization term is what people usually forget to add. The regularization term controls the complexity of the model, which helps us to avoid overfitting. This sounds a bit abstract, so let us consider the following problem in the following picture. You are asked to fit visually a step function given the input data points on the upper left corner of the image. Which solution among the three do you think is the best

fit?
The correct answer is marked in red. Please consider if this visually seems a reasonable fit to you. The general principle is we want both a simple and predictive model. The tradeoff between the two is also referred as bias-variance tradeoff in machine learning.

1.3 为什么介绍一般原则

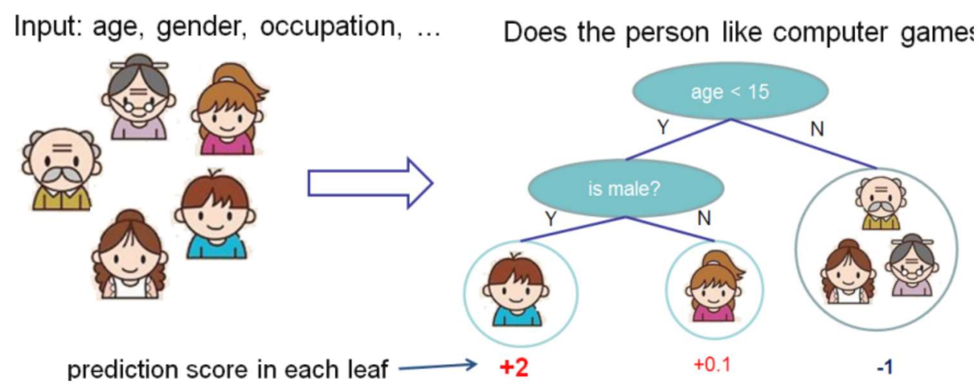
上面介绍的内容是有监督学习的基本要素，它们自然而然地成为了机器学习工具包的基本组成部分。例如，你应该能够描述梯度提升树和随机森林的相同点和不同点。以形式化方法理解这一过程，有助于我们理解正在学习的对象以及启发式算法背后的原因，例如剪枝和平滑。

Why introduce the general principle?

The elements introduced above form the basic elements of supervised learning, and they are natural building blocks of machine learning toolkits. For example, you should be able to describe the differences and commonalities between gradient boosted trees and random forests. Understanding the process in a formalized way also helps us to understand the objective that we are learning and the reason behind the heuristics such as pruning and smoothing.

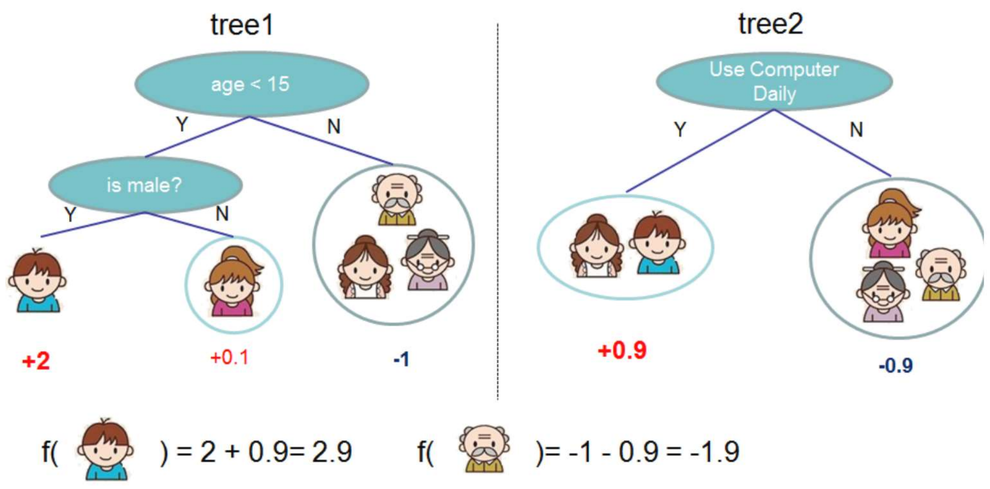
2 决策树集成

介绍完有监督学习的要素，现在开始进入真正的树模型。首先，了解一下XGBoost所选择的模型是：**决策树集成**（decision tree ensemble）。树集成模型包含了一组分类和回归树（CART）。下图展示了一个CART用于对“某人是否会喜欢电脑游戏”分类的简单例子。


Input: age, gender, occupation, …
Does the person like computer games
age < 15
Y
N
is male?
Y
N
prediction score in each leaf
+2
+0.1
-1

我们将一个家庭的成员划分入不同的叶子节点，并且给他们分配所在叶子节点相对应的分数。与叶子节点仅包含了决策值的决策树不同，在CART中，一个实数分值是与每个叶节点相关联的，这点比起分类器能让我们对结果有更深的理解。正如我们将在接下来的章节会看到的那样，这也使得能够从原理上、以更统一的方式去做优化。

通常而言，单棵树的能力是不够直接被应用在实践中的，实践中运用的是集成模型，即将多棵树的预测值求和。



这是一个由两棵树构成的树集成示例，最终得分由单棵树的预测分值求和得到。一个关键点是，这两个树所做的是试图互相**补充**。形式上我们可以用以下公式来表示模型：

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i),\ f_k \in F$$

式中K是树的总数，f是属于函数空间F的函数，F是所有可能的CART的集合。待优化的目标函数如下：

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

现在有个巧妙的问题：随机森林使用的是什么模型？是树集成！所以随机森林和提升树其实是同样的模型，不同之处在于我们如何训练它们。这就意味着，如果你想用树集成来写一个预测服务，只需要写一个就好了，它既能用随机森林又能用梯度提升树（参见Treelite的实例），一个说明为什么有监督学习的元素会抖动的例子。

Decision Tree Ensembles

Now that we have introduced the elements of supervised learning, let us get started with real trees. To begin with, let us first learn about the model choice of XGBoost: decision tree ensembles. The tree ensemble model consists of a set of classification and regression trees (CART). Here's a simple example of a CART that classifies whether someone will like computer games.
We classify the members of a family into different leaves, and assign them the score on the corresponding leaf. A CART is a bit different from decision trees, in which the leaf only contains decision values. In CART, a real score is associated with each of the leaves, which gives us richer interpretations that go beyond classification. This also allows for a pricipled, unified approach to optimization, as we will see in a later part of this tutorial.
Usually, a single tree is not strong enough to be used in practice. What is actually used is the ensemble model, which sums the prediction of multiple trees together.
Here is an example of a tree ensemble of two trees. The prediction scores of each individual tree are summed up to get the final score. If you look at the example, an important fact is that the two trees try to complement each other. Mathematically, we can write our model in the form

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i),\ f_k \in F$$

where K is the number of trees, f is a function in the functional space F, and F is the set of all possible CARTs. The objective function to be optimized is given by

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

Now here comes a trick question: what is the model used in random forests? Tree ensembles! So random forests and boosted trees are really the same models; the difference arises from how we train them. This means that, if you write a predictive service for tree ensembles, you only need to write one and it should work for both random forests and gradient boosted trees. (See Treelite for an actual example.) One example of why elements of supervised learning rock.

3 树提升

已经介绍完了模型，接下来我们把目光聚焦在训练上：我们要如何学习出这些树呢？答案是，正如所有有监督学习模型一直做的事：**定义一个目标函数然后做优化。**

现在假设下面这个函数是我们的目标函数（记得它始终需要含有训练损失和正则项）：

$$obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{k=1}^{t} \Omega(f_i)$$

Tree Boosting
Now that we introduced the model, let us turn to training: How should we learn the trees? The answer is, as is always for all supervised learning models: define an objective function and optimize it!
Let the following be the objective function (remember it always needs to contain training loss and regularization):

$$obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{k=1}^{t} \Omega(f_i)$$

3.1 累加训练

我们会想询问的第一个问题是：树的**参数**有哪些？你会发现我们需要学习得到的就是那些函数f_i，每个都包含了树结构及叶节点分数。学习树结构比传统的最优化问题难多了，在传统最优化问题中只需要简单地取梯度就好。一次性地学习出所有的树是很难解决的。作为替代，我们使用累加策略：保持已经得到的训练结果不变，仅仅在每次训练时增加一棵新的树。设第t步中得到的预测值是$\hat{y}_i^{(t)}$。那么我们有：

$$\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\dots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}$$

还有一个问题：在每步中加入的树要怎么选？一个很自然的想法就是加入那棵能够最优化我们的目标函数的树。

$$\begin{aligned}
obj^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{k=1}^{t} \Omega(f_i) \\
&= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant
\end{aligned}$$

如果考虑使用均方差（MSE）作为损失函数，那么目标就变为：

$$\begin{aligned}
obj^{(t)} &= \sum_{i=1}^{n} (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 + \sum_{i=1}^{t} \Omega(f_i) \\
&= \sum_{i=1}^{n} [2(\hat{y}_i^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2] + \Omega(f_t) + constant
\end{aligned}$$

MSE的公式非常友好，有一个一次性（通常称为残差）和一个二次项。如果使用其他损失函数（比如逻辑损失），就很难得到这么漂亮的公式了。因此在这个一般情况下，我们**对损失函数进行二次泰勒展开**：

$$obj^{(t)} = \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + constant$$

式中g_i和h_i的定义是：

$$\begin{aligned}
g_i &= \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \\
h_i &= \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})
\end{aligned}$$

移除所有常数项后，在第t步中的特定目标即为：

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

这就是我们构造新树的最优化目标。这一定义带来的一个重要优势是，目标函数值仅取决于g_i和h_i，这就是XGBoost能够支持自定义目标函数的原因。我们能够优化所有损失函数，包括逻辑回归和pairwise排序，对于不同的输入值g_i和h_i使用的是完全一致的解决方案。

Additive Training

The first question we want to ask: what are the parameters of trees? You can find that what we need to learn are those functions f_i, each containing the structure of the tree and the leaf scores. Learning tree structure is much harder than traditional optimization problem where you can simply take the gradient. It is intractable to learn all the trees at once. Instead, we use an additive strategy: fix what we have learned, and add one new tree at a time. We write the prediction value at step t as $\hat{y}_i^{(t)}$. Then we have

$$\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\dots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}$$

It remains to ask: which tree do we want at each step? A natural thing is to add the one that optimizes our objective.

$$\begin{aligned} obj^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{k=1}^{t} \Omega(f_i) \\ &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant \end{aligned}$$

If we consider using mean squared error (MSE) as our loss function, the objective becomes

$$\begin{aligned} obj^{(t)} &= \sum_{i=1}^{n} (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 + \sum_{i=1}^{t} \Omega(f_i) \\ &= \sum_{i=1}^{n} [2(\hat{y}_i^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2] + \Omega(f_t) + constant \end{aligned}$$

The form of MSE is friendly, with a first order term (usually called the residual) and a quadratic term. For other losses of interest (for example, logistic loss), it is not so easy to get such a nice form. So in the general case, we take the Taylor expansion of the loss function up to the second order:

$$obj^{(t)} = \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + constant$$

where the g_i and h_i are defined as

$$\begin{aligned} g_i &= \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \\ h_i &= \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \end{aligned}$$

After we remove all the constants, the specific objective at step t becomes

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

This becomes our optimization goal for the new tree. One important advantage of this definition is that the value of the objective function only depends on g_i and h_i. This is how XGBoost supports custom loss functions. We can optimize every loss function, including logistic regression and pairwise ranking, using exactly the same solver that takes g_i and h_i as input!

3.2 模型复杂度

介绍完了训练步骤，但是稍等，还有一件很重要的事情，那就是**正则项**！我们需要为树定义复杂度$\Omega(f)$。为了做这件事，让我们改进一下对于树$f(x)$的定义：

$$f_t(x) = \omega_{q(x)},\ \omega \in R^T,\ q : R^d \rightarrow \{1, 2, \ldots, T\}$$

这里 ω 是叶节点分值向量，q是将每个数据点分配到相对应的叶节点中去的函数，T是叶子数量。在XGBoost中，我们定义复杂度如下：

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

当然，还有其他多种方式来定义复杂度，但是上式的定义在实践中效果最好。正则项是被大多数树算法的包粗略对待甚至直接忽视的一部分内容。这是因为传统对于树的学习只重视改善不纯度，而对复杂度的控制就留给了启发式算法。通过形式化地定义正则项，我们能够对正在学习的东西有更好的认识，并得到一个更加泛化的模型。

Model Complexity

We have introduced the training step, but wait, there is one important thing, the regularization term! We need to define the complexity of the tree Ω(f). In order to do so, let us first refine the definition of the tree f(x) as

$$f_t(x) = \omega_{q(x)},\ \omega \in R^T,\ q : R^d \rightarrow \{1, 2, \ldots, T\}$$

Here w is the vector of scores on leaves, q is a function assigning each data point to the corresponding leaf, and T is the number of leaves. In XGBoost, we define the complexity as

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

Of course, there is more than one way to define the complexity, but this one works well in practice. The regularization is one part most tree packages treat less carefully, or simply ignore. This was because the traditional treatment of tree learning only emphasized improving impurity, while the complexity control was left to heuristics. By defining it formally, we can get a better idea of what we are learning and obtain models that perform well in the wild.

3.3 结构分数

导数中有一个神奇的部分。在重构树模型后，我们可以将第t棵树的目标值写作：

$$\begin{aligned} obj^{(t)} &\approx \sum_{i=1}^{n}[g_i\omega_{q(x_i)} + \frac{1}{2}h_i\omega_{q(x_i)}^2] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T}\omega_j^2 \\ &= \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)\omega_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)\omega_j^2] + \gamma T \end{aligned}$$

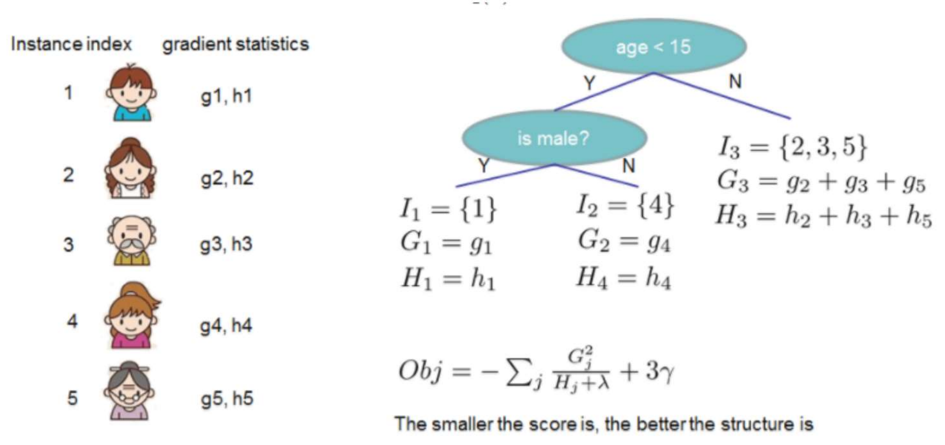
式中 $I_j = \{i \mid q(x_i) = j\}$ 是被分配到第j个叶子的数据点的下标集合。注意到在第二行中，我们变换了求和函数的下标，因为位于同个叶子中的所有数据点得到的分值是相同的。更近一步化简表达式，定义 $G_j = \sum_{i\in I_j} g_i$，$H_j = \sum_{i\in I_j} h_i$，得到：

$$obj^{(t)} = \sum_{j=1}^{T}[G_j\omega_j + \frac{1}{2}(H_j + \lambda)\omega_j^2] + \gamma T$$

在这一等式中，w_j是互相独立的，式子$G_j\omega_j + \frac{1}{2}(H_j + \lambda)\omega_j^2$是一个二次项。对于给定的结构$q(x)$，使目标函数的最小化的$\omega_j$的取值、及最小化的目标函数为：

$$\begin{aligned} \omega_j^* &= -\frac{G_j}{H_j + \lambda} \\ obj^* &= -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T \end{aligned}$$

后一个等式衡量了树结构$q(x)$有**多好**。



如果所有这些听起来有点复杂，让我们看看在这张图片中，分数是怎样计算得出的。总的来说，对于一个给定的树结构，我们将定值g_i和h_i放到它们对应的叶节点中，将这些值求和，然后运用公式计算出这棵树有多好。这个得分很像决策树中的不纯度，只是它将模型复杂度也考虑进去了。

The Structure Score

Here is the magical part of the derivation. After re-formulating the tree model, we can write the objective value with the t-th tree as:

$$\begin{aligned} obj^{(t)} &\approx \sum_{i=1}^{n}[g_i\omega_{q(xi)} + \frac{1}{2}h_i\omega_{q(xi)}^2] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T}\omega_j^2 \\ &= \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)\omega_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)\omega_j^2] + \gamma T \end{aligned}$$

where $I_j = \{i|q(xi) = j\}$ is the set of indices of data points assigned to the j-th leaf. Notice that in the second line we have changed the index of the summation because all the data points on the same leaf get the same score. We could further compress the expression by defining $G_j = \sum_{i\in I_j} g_i$ and $H_j = \sum_{i\in I_j} h_i$:

关注我
0 0

$$obj^{(t)} = \sum_{j=1}^{T}[G_j\omega_j + \frac{1}{2}(H_j+\lambda)\omega_j^2] + \gamma T$$

In this equation, w_j are independent with respect to each other, the form $G_j\omega_j + \frac{1}{2}(H_j+\lambda)\omega_j^2$ is quadratic and the best wj for a given structure q(x) and the best objective reduction we can get is:

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

The last equation measures how good a tree structure q(x) is.
If all this sounds a bit complicated, let’s take a look at the picture, and see how the scores can be calculated. Basically, for a given tree structure, we push the statistics gi
and hi to the leaves they belong to, sum the statistics together, and use the formula to calculate how good the tree is. This score is like the impurity measure in a decision tree, except that it also takes the model complexity into account.

3.4 学习树结构

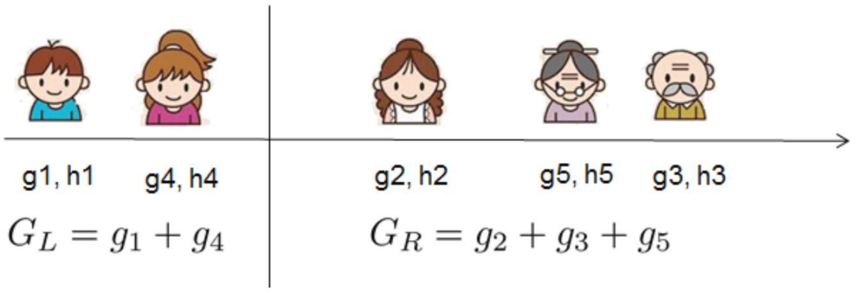
既然我们已有了衡量一棵树有多好的方法，理论上我们可以枚举所有可能的树然后挑出最好的，但在实际中这是很难做到的，所以我们将尝试每次优化树的一层。

特别地，若我们试图将一个叶节点划分为两个叶节点，此时分数增益为：

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

这个公式可被拆分为 1)左子树得分 2）右子树得分 3) 原叶节点得分 4）对新增添的叶子的正则化项。显而易见的是，如果增益值比γ值小，我们就最好不要添加这个分支。这也正是所有基于树的模型的**剪枝**技术。通过运用有监督学习的原则，我们自然能想到这些技术能够起效果的原因 :)

对于实数值的数据，我们通常想要找到最优的切分点。为了高效地做这件事，我们先将所有样本排序好，像下图所示：



只要从左到右扫描，就足够用于计算所有可能的切分方案的结构分数，然后我们就能高效地找到最佳切分点。

Learn the tree structure

Now that we have a way to measure how good a tree is, ideally we would enumerate all possible trees and pick the best one. In practice this is intractable, so we will try to optimize one level of the tree at a time.
Specifically we try to split a leaf into two leaves, and the score it gains is

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

This formula can be decomposed as 1) the score on the new left leaf 2) the score on the new right leaf 3) The score on the original leaf 4) regularization on the additional leaf. We can see an important fact here: if the gain is smaller than γ , we would do better not to add that branch. This is exactly the pruning techniques in tree based models! By using the principles of supervised learning, we can naturally come up with the reason these techniques work :)
For real valued data, we usually want to search for an optimal split. To efficiently do so, we place all the instances in sorted order, like the following picture.
A left to right scan is sufficient to calculate the structure score of all possible split solutions, and we can find the best split efficiently.

关注我
0 0

关于公式推导中常数项的解释：在第t步时，前t-1步的运算结果都可视作已知（常数）。

原文：Introduction to Boosted Trees

相关概念阅读参考：
Understanding the Bias-Variance Tradeoff
分类与回归树（Classification and Regression Trees, CART）
L1, L2 Regularization – Why needed/What it does/How it helps?

感谢南大薛恺丰同学帮忙校对~

分类: Machine Learning