XGBoost 学习与实现

汪加林

2016 年 12 月 10 日

1 Introduction

理论相对 SVM 而言比较简单，本文也会推导一遍，然后再讲解代码的实现。
下面是整个文档的组织内容。

- XGBoost 的理论推导
- 一些有意思的问题
- 实验
- 类的设计

2 XGBoost 的理论推导

2.1 GBDT

首先来看看一般 GBDT(Gradient Boosting Decision Tree) 的算法。

可以看到，正如其名字所示，和梯度下降法非常类似，不断的减去其梯度。直至梯度近似为 0，就使 $L(y, F)$ 最小。GDBT 采用的是一阶导数信息。

2.2 XGBoost

定义目标优化函数 $Obj = L(y, F_K) + \Omega(F_k)$。$F_k$ 表示学习到的树，前者表示经验损失，后者表示正则项。假设每次学到的树为 $f_t(x) = w_{q(x)}$,w 表示树的叶子节点的权重,$q(x)$ 表示树的结构。如下图所示：

Algorithm 1 Gradient Boosting Decision Tree

Input:

$\{x, y\}_1^N$

Output:

$F_{numTrees}$

$F_0 = 0$

for $t = 1, 2, ..., numTrees$ **do**

$\hat{r_t} = \frac{\partial(L(y, F_{t-1}))}{\partial(F_{t-1})}$

$\{w_i, q(x)_i\}_1^J = argmin_{w,q}[\hat{r_t} - f_t(w, q))]^2$

$p = argmin_p L(y, F_{t-1} + p * f_t)$

$f_t = p * f_t$

$F_t = F_{t-1} + f_t$

end for

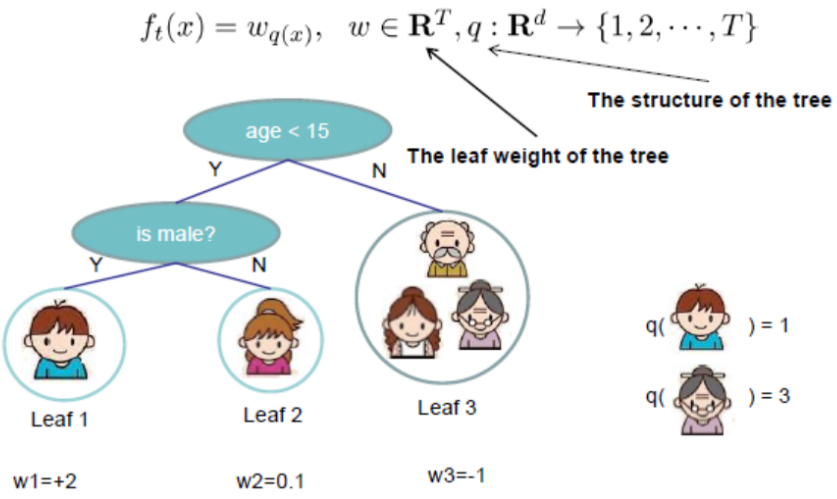

Figure 1: 各符号的图解说明

$$
\begin{aligned}
Obj &= L(y, F_k) + \Omega(F_k) \\
&= \sum_{i=1}^{N} L(y_i, (F_{k-1} + f_k)_i) + \Omega(F_{k-1}) + \Omega(f_k) \\
&\approx \sum_{i=1}^{N} L(y_i, (F_{k-1} + f_k)_i) + \frac{\partial L(y, F_{k-1})}{\partial F_{k-1}} * f_k + \frac{1}{2}\frac{\partial^2 L(y, F_{k-1})}{\partial F_{k-1}^2} * f_k + \Omega(F_{k-1}) + \Omega(f_k) \\
&= \sum_{i=1}^{N} \{\frac{\partial L(y_i, (F_{k-1} + f_k)_i)}{\partial F_{k-1}} * f_k + \frac{1}{2}\frac{\partial^2 L(y_i, (F_{k-1} + f_k)_i)}{\partial F_{k-1}^2} * f_k^2\} + \Omega(f_k) + constant
\end{aligned}
$$

上式中近似的那一步，用到了泰勒展开式。$L(y_i, (F_{k-1} + f_k)_i)$ 看成是 F_{k-1} 的函数，泰勒展开在 F_{k-1} 处的展开，$\Delta x = f_k$。

$$
\begin{aligned}
g_i &= \frac{\partial L(y_i, (F_{k-1} + f_k)_i)}{\partial F_{k-1}} \\
h_i &= \frac{\partial^2 L(y_i, (F_{k-1} + f_k)_i)}{\partial F_{k-1}^2}
\end{aligned}
$$

$$
Obj = \sum_{i=1}^{N} (g_i f_k + \frac{1}{2} h_i f_k^2) + \Omega(f_k) + constant
$$

令 $\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \sum_{i=1}^{T} w_i^2$

因此：

$$
\begin{aligned}
Obj &= \sum_{i=1}^{N} (g_i f_k + \frac{1}{2} h_i f_k^2) + \Omega(f_k) + constant \\
&= \sum_{i=1}^{N} (g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2) + \gamma T + \frac{1}{2}\lambda \sum_{i=1}^{T} w_i^2 + constant \\
&= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) * w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T
\end{aligned}
$$

令 $G_j = \sum_{i \in I_j} g_i$,$H_j = \sum_{i \in I_j} h_i$

$$
Obj = \sum_{j=1}^{T} \{G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2\} + \gamma T
$$

当树的结构确定的时候 (i.e.G_j, H_j 是确定的)，目标函数是 w_j 的二次函数。当取到最优点的时候，$w_j = -\frac{G_j}{H_j+\lambda}$，

$$Obj_{min} = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T + constant$$

那么如何确定树的结构，采用穷举的方法复杂度太大了。最终采用了贪心算法。每次分裂时选取增益最大（使目标函数最小）的分裂特征和相应的特征值。每次分裂后增益：假设对于某棵树的某节点 j，$G_j = G$，并且假设分裂后的左子节点取值为 G_L H_L, 右子节点取值为 G_R, H_R, 并且 $G = G_L + G_R, H = H_L + H_R$。

$$\begin{aligned} Gain &= Obj_{old} - Obj_{new} \\ &= -\frac{1}{2}\frac{G^2}{H+\lambda} + \gamma T - \{-\frac{1}{2}((\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda}) + \gamma(T+1)\} \\ &= \frac{1}{2}(\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{(H_L+H_R)+\lambda}) - \gamma \end{aligned}$$

那么下一个问题便是，如何寻找最佳的分裂点呢？比较直观的是对每一个特征，将该节点的所有样本的该特征值从小到大排序，从小到大遍历，选择增益 $Gain$ 最大的那个特征和对应的特征值分裂。这样的话便只用对于排好序的样本，便只用按顺序从左至右扫描就好了。

因此整个的 XGBoost 算法流程如下图所示。

3 一些有意思的问题

推导 xgboost 的过程其实就是一个优化的过程，和我们之前推导 svm 的想法是非常相似的。目标函数都是经验损失函数和正则化两项组成的。

- 优化经验损失函数是为了让我们的模型能很好的描述当前的训练数据；
- 优化正则项则是为了让我们的模型看起来更简单，防止过拟合，具有更强的泛化能力。

那么在 xgboost 的实践中，有哪些方法可以控制过拟合呢？大家如果看过官方 xgboost 的 API 就会发现，有很多方式可以控制：

- 控制树的复杂度，通常是树的深度；
- 采样，每次训练只使用部分的数据

Algorithm 2 节点分裂算法

Input:

$\{x, y\}_1^{DataNum}$: 该节点的数据样本

g：按照理论计算的一阶倒数

h: 按照理论计算的二阶倒数

$Gain = 0$

for $i = 1, 2, ..., FeatueNum$ **do**

sortedId = sort(x[:][i]) //返回的是排序的索引

$G_L = 0, H_L = 0$

for j=1,2,...,DataNum **do**

$G_L = G_L + g_{sortedId[j]}$

$G_R = sum(g) - G_L$

$H_L = H_L + h_{sortedId[j]}$

$H_R = sum(h) - H_L$

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

记录 $Gain$ 最大时的分裂特征和最佳特征值

end for

end for

if Gain>0 **then**

按照上述记录的值进行分裂

end if

Algorithm 3 XGBoost

Input:

$\{x, y\}_1^N$

Output:

$F_{numTrees}$

$F_0 = 0$

for $t = 1, 2, ..., numTrees$ **do**

利用节点分裂算法生成树 f_t

$F_t = F_{t-1} + \epsilon f_t$ //这里

end for

- 列采样，训练时，只使用部分特征
- shrinkage：给每棵树的叶子权重进行缩减，给后面的训练预留空间
- Early Stop：根据测试集的情况选用前若干树

参考：[1] 腾讯的火光摇曳，

http://www.flickering.cn/machine_learning/2016/08/gbdt详解上-理论/

[2] 陈天奇大神的 xgboost PPT。

4 实验

使用的是 heart_scale 数据，二分类。$\gamma = 10; \lambda = 2; minimum = 3; numTrees = 500$; 训练时间为: 557, 预测正确率为：0.742。

5 类的设计

根据整个算法的流程，设计了以下的类:

Node : 表示节点

Tree ：树的结构的表示

Loss : 损失函数的选择，未实现，目前采用的是 Square Loss。

TwoDimE ：表示一个二元数组，

Queue ：存放树分裂过程中的节点

SimplifiedXGBoost : 控制整个算法流程的实现

GrowTree ：实现树的生成算法

PostPrune : 后剪枝，未实现，目前采用的是预剪枝。

Predict ：实现样本的预测

TwoDimComp ：二元数组的 Comparator

Utils ：数据的读取