深入理解Logistic Loss与回归树

Logistic Function与Logistic Regression

Logistic Function最常见的定义形式如下：

$$P(x) = \frac{1}{1+e^{-x}} \tag{1}$$

其中$x \in R$，$P(x) \in [0,1]$，实际上这个公式起源于伯努利分布，$P(x)$代表概率，关于其起源在此暂不赘述。

公式(1)有一个重要的性质，即：

$$\begin{aligned} P(-x) &= \frac{1}{1+e^{x}} \\ &= \frac{1}{1+\frac{1}{e^{-x}}} \\ &= \frac{e^{-x}}{1+e^{-x}} \\ &= \frac{e^{-x}+1-1}{1+e^{-x}} \\ &= 1-\frac{1}{1+e^{-x}} \\ &= 1-P(x) \end{aligned} \tag{2}$$

并且公式(1)的形式在逻辑回归二分类中被广泛应用，其中P通常代表预测样本属于正例的概率。假设$\vec{x}$代表由多个特征组成的样本向量，$\vec{\beta}$代表由每个特征的系数构成的向量，y代表样本的类别标记，$y=1$代表正例，$y=0$代表反例，则

$$\begin{aligned} P(y=1|\vec{\beta},\vec{x}) &= \frac{1}{1+e^{-\vec{\beta}^T\vec{x}}} \\ &= \frac{e^{\vec{\beta}\vec{x}}}{1+e^{\vec{\beta}\vec{x}}} \end{aligned} \tag{3}$$

$$\begin{aligned} P(y=0|\vec{\beta},\vec{x}) &= 1-P(y=1|\vec{\beta},\vec{x}) \\ &= \frac{1}{1+e^{\vec{\beta}\vec{x}}} \end{aligned} \tag{4}$$

如果反例标记用-1表示，则预测正例样本的公式和预测反例样本的公式可以集成为一个公式：

$$P(g=\pm 1|\vec{\beta},\vec{x}) = \frac{1}{1+e^{-g\vec{\beta}\vec{x}}} \tag{5}$$

其中$g=1$代表正例，$g=-1$代表反例。

二进制交叉熵与Logistic Loss

假设f是hypothesis function，而L是loss function，训练有监督学习器的时候实际上都是如下的优化问题：

$$arg\min \sum_i L(y_i, f(\vec{x}_i)) \tag{6}$$

我们已经知道如果是二分类器的话，且输出是概率形式，那么我们可以选择二进制交叉熵来作为loss function，比如假设f就是logistic function

$$f(\vec{x}) = \frac{1}{1+e^{-\vec{\beta}\vec{x}}} \tag{7}$$

用二进制交叉熵作为loss function，正例标记为1，反例标记为0，则

$$\begin{aligned} L(y_i, f(\vec{x}_i)) &= -[y_i log(f(\vec{x}_i)) + (1-y_i)log(1-f(\vec{x}_i))] \\ &= -[y_i log(\frac{1}{1+e^{-\vec{\beta}\vec{x}_i}}) + (1-y_i)log(1-\frac{1}{1+e^{-\vec{\beta}\vec{x}_i}})] \\ &= y_i log(1+e^{-\vec{\beta}\vec{x}_i}) - (1-y_i)log\frac{e^{-\vec{\beta}\vec{x}_i}}{1+e^{-\vec{\beta}\vec{x}_i}} \\ &= y_i log(1+e^{-\vec{\beta}\vec{x}_i}) - (1-y_i)log\frac{1}{1+e^{\vec{\beta}\vec{x}_i}} \\ &= y_i log(1+e^{-\vec{\beta}\vec{x}_i}) + (1-y_i)log(1+e^{\vec{\beta}\vec{x}_i}) \end{aligned} \tag{8}$$

假设h是另外一个hypothesis function，且$h = \vec{\beta}\vec{x}$，把h带入(8)式可以得到

$$L(y_i, h(\vec{x}_i)) = y_i log(1+e^{-h(\vec{x}_i)}) + (1-y_i)log(1+e^{h(\vec{x}_i)}) \tag{9}$$

显然(8)式和(9)式是等价的，即

$$L(y_i, f(\vec{x}_i)) = L(y_i, h(\vec{x}_i)) \tag{10}$$

(9)式的Loss形式叫作针对h的Logistic Loss，也就是说针对f的二进制交叉熵损失等价于针对h的Logistic损失，而f实际上是h通过Logistic function的映射，即

$$f(\vec{x}) = \frac{1}{1+e^{-h(\vec{x})}} \tag{11}$$

也就是说不论h这个hypothesis function是什么形式，针对h使用Logistic Loss就可以用于二分类问题，因为针对h使用Logistic Loss就等价于针对f使用二进制交叉熵损失。

如果是用于二分类的深层神经网络，这里的f则可以看成是神经网络的最后一个Sigmoid输出神经元，如果$\vec{x}$仍然看作是原始输入样本的话，则h可以看成是从输入层到最后一个隐层整体构成的复合函数。

著名的梯度提升机(GBM, Gradient Boosting Machine)实际上可以对任何学习器进行Boosting操作，Boosting是一种集成学习方法，已经证明任何弱学习器都有提升为强学习器的潜能，因此Boosting通常指把弱学习器提升为强学习器的提升方法。另外，这里的Gradient是指Gradient Descent，不是中文字面意思的“梯度值提升”，“提升”是指Boosting这种集成学习范式。

GBM最常见的是GBDT(Gradient Boosting Decision Tree)，因为GBDT实际上用的是基于CART中的回归树的加性模型，所以GBDT也常叫作GBRT(Gradient Boosting Regression Tree)，也就是说GBDT中的弱学习器是经典的Breiman的CART中的Regression Tree，我们知道GBDT既可以用于回归也可以用于分类，那么回归树是如何用于分类的呢?

这里说明二分类的情况，多分类情况暂不赘述。由(11)式知道如果h就是回归树的hypothesis function，也就是说h就是回归树的叶子结点的输出，那么通过Logistic function映射成f后再对f使用二进制交叉熵损失就可以做二分类任务了，由$(9)(10)$式知道也就是说直接对回归树h使用Logistic Loss就可以做二分类任务了。

CART回归树的回归与分类数学原理

CART(Classification and Regression Tree)是最早由Breiman等人提出的一类决策树算法，其中Classification Tree使用基尼系数作为分裂准则，在此暂不赘述分类树细节，Regression Tree采用启发式算法寻找最优分裂节点，这里推导一下Loss function分别采用Squared Loss和Logistic Loss的回归和分类的数学原理。

采用Squared Loss的回归树回归

关于CART回归树的原理推导可以参考李航的《统计学习方法》，我这里试图用更通俗易懂，更详细严谨的方式进行解读和推导。

首先假设hypothesis function为f，也即回归树为f，样本向量为$\vec{x}$，$y \in R$是样本$\vec{x}$的真实target值，则回归树f的回归预测公式为

$$\hat{y}_i = f(\vec{x}_i) \tag{12}$$

其中i为样本编号，$\hat{y}_i$为回归树第i个样本的估计值。

Squared Loss形式记为

$$L_{squared}(y_i, f(\vec{x}_i)) = (y_i - f(\vec{x}_i))^2 \tag{13}$$

其中y_i为样本编号为i的真实target值。

CART回归树为二叉树，其关键在于如何在特征空间中选择最优分裂节点进行多次二分叉分裂。用表格的形式更容易进行通俗易懂的解释，为了严谨这里先对张量的概念做一个通俗易懂的理解。

通俗理解张量：向量是一维张量，矩阵(表格)是二维张量，而像魔方那样每个小方块代表一个数据，所有数据存储在一个“三维”空间中的是三维张量(比如三通道的RGB彩色图片)，当然还有更高维的张量，在各种编程语言中就对应着高维数组，实际上在物理学中对张量有严格的定义，由于其概念过于抽象，这里暂不赘述。

现在假设训练数据都存放在一张二维表格中，这里的“二维”指张量的二维。假设表格有$d+1$列，N行，前d列为特征，最后一列为target y，每一行的前d列构成d维样本向量$\vec{x}$，前d列所有数据叫作特征空间(输入空间)，最后一列叫作输出空间。回归树用启发式方法在特征空间中寻找最优分裂节点对表格进行多次划分，意思就是遍历每个特征的取值，根据每个特征的取值作为判断依据(分裂节点)把表格一分为二，一分为二后的两份表格的这个特征的所有取值要么都大于等于这个判断依据(分裂节点)，要么都小于等于这个判断依据(分裂节点)。第一次把原始表格进行一分为二划分的节点叫作根(root)节点，后面用同样的方法把“子表格”进行划分的节点都叫中间节点。那么如何判断哪个特征的哪个值是当前的最优分裂节点?

我们知道原始表格最终一定会被划分成多个子表格，每个子表格实际上是由多个判据值(分裂节点)所构成的规则下的样本的集合，这些样本同属一类，高度相似，因此可以假设每个子表格里(也就是符合这套规则的所有样本)的所有样本对应同一个估计值w，假设最终总子表格数为M，用$T_1, T_2, \cdots, T_m, \cdots, T_M$代表相应的子表格(样本集合)，相应的估计值就为$w_1, w_2, \cdots, w_m, \cdots, w_M$，也即

$$f(\vec{x}_i | \vec{x}_i \in T_m) = w_m \tag{14}$$

也就是说回归树的hypothesis function为

$$\begin{aligned} f(\vec{x}_i) &= \sum_{m=1}^{M} f(\vec{x}_i | \vec{x}_i \in T_m) I(\vec{x}_i \in T_m) \\ &= \sum_{m=1}^{M} w_m I(\vec{x}_i \in T_m) \end{aligned} \tag{15}$$

其中

$$I(\vec{x}_i) = \begin{cases} 1 & \vec{x}_i \in T_m \\ 0 & \vec{x}_i \notin T_m \end{cases} \tag{16}$$

根据(13)式最小化Squared Loss

$$\begin{aligned} arg\min \sum_{i=1}^{N} L_{squared}(y_i, f(\vec{x}_i)) &= arg\min \sum_{i=1}^{N} (y_i - f(\vec{x}_i))^2 \\ &= arg\min \sum_{i=1}^{N} (y_i - \sum_{m=1}^{M} w_m I(\vec{x}_i \in T_m))^2 \\ &= arg\min \sum_{m=1}^{M} \sum_{\vec{x}_i \in T_m} (y_i - w_m)^2 \end{aligned} \tag{17}$$

$$\begin{aligned} \frac{\partial}{\partial w_m} \sum_{i=1}^{N} L_{squared}(y_i, f(\vec{x}_i)) &= \sum_{m=1}^{M} \frac{\partial}{\partial w_m} \sum_{\vec{x}_i \in T_m} (y_i - w_m)^2 \\ &= -2 \sum_{\vec{x}_i \in T_m} (y_i - w_m) \end{aligned} \tag{18}$$

令(18)式等于0，有

$$-2 \sum_{\vec{x}_i \in T_m} (y_i - w_m) = 0 \tag{19}$$

$$\Rightarrow \sum_{\vec{x}_i \in T_m} w_m = \sum_{\vec{x}_i \in T_m} y_i \tag{20}$$

$$\Rightarrow w_m = ave(y_i | \vec{x}_i \in T_m) \tag{21}$$

由(21)式知道在Squared Loss下每个T_m内部的最优估计值就是T_m内部所有样本对应y_i的平均值。那么每一次分裂要如何选择最优分裂节点并一步步迭代呢?

我们已经知道每一次分裂相当于把表格一分为二，并且已经知道了在Squared Loss下分裂之后每个子表格对应的估计值取子表格内部所有样本对应的target值的平均值就可以使得子表格内部的Squared Loss最小，也就是说只要每次分裂后的两个子表格的最小Squared Loss值相加最小即可，采用的方法就是遍历每个特征及其每一个取值，尝试把每一个特征下的具体的值作为分裂节点，分裂成两个子表格后计算两个子表格的最小Squared Loss的和，如果这个和在遍历了所有可能的分裂节点后是最小的，那么这一次分裂就选当前的分裂节点。

具体地，假设$x^{(j)}$是第j个特征，s是特征$x^{(j)}$的某个取值，把$x^{(j)}$和s分别作为切分变量和切分点，则可以把当前表格切分成如下两个表格：

$$T_1(j, s) = \{x | x^{(j)} \leq s\} \tag{22}$$

和

$$T_2(j, s) = \{x | x^{(j)} > s\} \tag{23}$$

计算下式寻找最优切分变量$x^{(j)}$和最优切分点s，

$$\min_{j,s} \left[\min_{w_1} \sum_{\vec{x}_i \in T_1} (y_i - w_1)^2 + \min_{w_2} \sum_{\vec{x}_i \in T_2} (y_i - w_2)^2 \right] \tag{24}$$

由(21)式可以知道$w_1 = ave(y_i | \vec{x}_i \in T_1)$和$w_2 = ave(y_i | \vec{x}_i \in T_2)$时可以使得(24)式在当前分裂点最小，也就是说只要遍历所有可能的分裂点(j, s)，找到全局最小的(j, s)即可。后续每一次表格的一分为二都采取同样的方法，直到满足停止条件。采用Squared Loss的回归树也叫最小二乘回归树。

采用Logistic Loss的回归树二分类

由(7)(8)式已经知道了Logistic Loss由Logistic function和Binary Entropy Loss推导而来，而Logistic function也叫Sigmoid function，所以Logistic Loss也叫Sigmoid Binary Entropy Loss，它可以把回归模型用于二分类任务。

Logistic Loss最常用的形式是(9)式的label $y \in \{0,1\}$的形式，如果label $y \in \{-1,1\}$，则其形态为

$$L(y_i, h(\vec{x}_i)) = log(1 + e^{-y_i h(\vec{x}_i)}) \tag{25}$$

在(15)式中已经知道了回归树的hypothesis function，这里尝试采用Logistic Loss作为loss function来推导其训练过程(这里纯属个人推导理解，真实的各种GBDT源码中的回归树不一定是这么实现的)。以下推导Logistic Loss采用(9)式的形态，所以取label $y \in \{0,1\}$。那么训练分类器也就是如下的优化问题

$$arg\min_{w_m} \sum_{i=1}^{N} L_{logistic}(y_i, f(\vec{x}_i)) = arg\min_{w_m} \sum_{i=1}^{N} \Big[y_i log(1 + e^{-f(\vec{x}_i)}) + (1 - y_i) log(1 + e^{f(\vec{x}_i)}) \Big] \tag{26}$$

$$\begin{aligned} \frac{\partial}{\partial w_m} \sum_{i=1}^{N} L_{logistic}(y_i, f(\vec{x}_i)) &= \frac{\partial}{\partial w_m} \sum_{m=1}^{M} \sum_{\vec{x}_i \in T_m} \big[y_i log(1 + e^{-w_m}) + (1 - y_i) log(1 + e^{w_m}) \big] \\ &= \sum_{\vec{x}_i \in T_m} \left[y_i \frac{-e^{-w_m}}{1 + e^{-w_m}} + (1 - y_i) \frac{e^{w_m}}{1 + e^{w_m}} \right] \\ &= \sum_{\vec{x} \in T_m} \left[y_i \frac{-e^{-w_m}}{1 + e^{-w_m}} + (1 - y_i) \frac{1}{1 + e^{-w_m}} \right] \\ &= \sum_{\vec{x} \in T_m} \left[\frac{1}{1 + e^{-w_m}} - y_i \right] \end{aligned} \tag{27}$$

令(27)式等于0，则

$$\sum_{\vec{x}_i \in T_m} (\frac{1}{1 + e^{-w_m}} - y_i) = 0 \tag{28}$$

$$\Rightarrow \sum_{\vec{x}_i \in T_m} \frac{1}{1 + e^{-w_m}} = \sum_{\vec{x}_i \in T_m} y_i \tag{29}$$

$$\Rightarrow \frac{1}{1 + e^{-w_m}} = ave(y_i | \vec{x}_i \in T_m) \tag{30}$$

$$\Rightarrow w_m = -log(\frac{1}{ave(y_i | \vec{x}_i \in T_m)} - 1) \tag{31}$$

那么分裂思想也就可以与前述的最小二乘回归树一样，只不过Loss采用Logistic loss，w_m采用(31)式的计算值。实际上由(30)式已经可以看出回归树的每个叶节点的权值都输入Logistic function所构成的级联模型在损失为二进制交叉熵下的模型输出值就是T_m里面y_i的均值。