LambdaMART Overview

Liam Huang*

2016 年 7 月 10 日

*liamhuang0205@gmail.com

1 简介

- Ranking 问题是搜索引擎需要解决的核心问题之一.
- 利用机器学习(Machine Learning)解决 Ranking 问题的算法叫做 Learning to Rank 算法，简称 LTR.
- LambdaMART 是 LTR 算法中的一种，属于 LTR 中的 listwise 算法.

LambdaMART 由两部分组成.

- Lambda 描述机器学习过程中迭代的方向和强度，来自 LambdaRank.
- MART 是 Multiple Additive Regression Trees 的缩写，实际就是 GBRT (Gradient Boosted Regression Trees) 也就是 GBDT (Gradient Boosting Decision Tree).

表 1: 符号说明

符号	说明
q	查询词
x	需要排序的某条文档
D	由全体 x 组成的集合
s	文档的得分
(i, j)	两个文档 x_i 和 x_j 组成的有序对（pair）
P	由全体上述 pair 组成的集合
$x_i \rhd x_j$	x_i 比 x_j 相关性更好
P_{ij}	x_i 排在 x_j 之前的预测概率
$\bar{P}_{ij}$	x_i 排在 x_j 之前的真实概率
$S_{ij} \in \{0, \pm 1\}$	描述 x_i 和 x_j 的相关性关系

2 MART

MART 的核心思想是 Boosting：*将一群弱学习器组合起来变成一个强学习器.*

- 弱学习器：深度有限的决策树；
- 强学习器：深度有限的决策树组成的森林.

目标：寻找一个强学习器 $f(x)$ 满足

$$\hat{f}(x) = \arg\min_{f(x)} E\Big[L\big(y, f(x)\big)\Big|x\Big].$$

强学习器由弱学习器组合起来：

$$\hat{f}(x) = \sum_{m=1}^{M} f_m(x).$$

$$\begin{aligned}\hat{f}_1 &= f_1 \\ \hat{f}_2 &= f_1 + f_2 \\ &\vdots \\ \hat{f}_M &= \sum_{m=1}^{M} f_m\end{aligned}$$

MART 每一步迭代的目标：尽可能使得损失下降. 对于 x_i 来说，就是要消灭

$$g_{im} = \frac{\partial L(y_i, \hat{f}_{m-1}(x_i))}{\partial \hat{f}_{m-1}(x_i)}.$$

因此可以定义 $f_m = -\rho_m g_m$，也就是说 MART 实际上是在泛函空间上做梯度下降.

决策树实际上将样本空间划分成了若干区域，并对每个划分区域赋上预测值．假设 f_m 划分而成的区域是 R_m，预测值则是 γ_m．那么，我们有

$$f_m(x) = h_m(x; R_m, \gamma_m),$$

也就是

$$\hat{f}(x) = \sum_{m=1}^{M} f_m(x) = \sum_{m=1}^{M} h_m(x; R_m, \gamma_m).$$

那么，MART 的每一步也就是要解优化问题：

$$h_m(x; R_m, \gamma_m) = \arg\min_{R,\gamma} \sum_{i=1}^{N} \big(-g_{im} - F(x_i; R, \gamma)\big)^2, \qquad (\bigstar)$$

其中 F 是一棵回归决策树．

MART 实际上是一个框架，因为最关键的迭代步骤中的 g_{im} 可以替换成任何能够实现「梯度」的函数．

3 Lambda

- RankNet
- LambdaRank
- LambdaMART

3.1 RankNet 的创新

Ranking 的常见评价指标都无法求梯度

- NDCG;
- ERR;
- MAP;
- MRR.

RankNet 将不适宜用梯度下降求解的问题，转化为对概率的交叉熵损失函数的优化问题，从而适用梯度下降方法.

RankNet 的终极目标是得到一个算分函数（的参数 w）

$$s = f(x; w).$$

根据这个算分函数，可以计算文档 x_i，x_j 的得分 s_i，s_j

$$s_i = f(x_i; w) \quad s_j = f(x_j; w),$$

再计算二者的偏序概率

$$P_{ij} = P(x_i \rhd x_j) = \frac{\exp\big(\sigma(s_i - s_j)\big)}{1 + \exp\big(\sigma(s_i - s_j)\big)} = \frac{1}{1 + \exp\big(-\sigma(s_i - s_j)\big)},$$

再以交叉熵为损失函数

$$L_{ij} = -\bar{P}_{ij} \log P_{ij} - (1 - \bar{P}_{ij}) \log(1 - P_{ij}) = \frac{1}{2}(1 - S_{ij})\sigma(s_i - s_j) + \log\Big\{1 + \exp\big(-\sigma(s_i - s_j)\big)\Big\},$$

进行梯度下降

$$w_k \to w_k - \eta \frac{\partial L}{\partial w_k}.$$

3.2 再探梯度

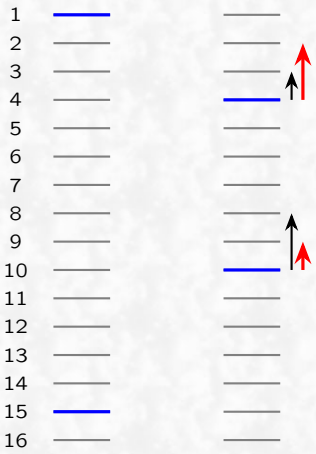

图 1: 梯度示意图

- RankNet 的梯度下降表现在结果整体变化中，是逆序对减少（$13 \to 11$）；
- RankNet 的梯度下降表现在单条结果的变化中，是结果在列表中的移动趋势（黑色箭头）；
- NDCG 关注 topK 的结果排序情况，我们希望结果的移动趋势如红色箭头所示.

那么可不可以直接定义梯度呢?

3.3 LambdaRank

- RankNet 告诉我们如何绕开 NDCG 得到一个可用的梯度;
- 上一节我们明确了我们需要怎样的梯度.

元芳，你怎么看?

- 红色箭头反映的是某条结果排序的变化;
- 排序是由模型得分 s 确定的;
- 元芳：我要扼住 $\frac{\partial L_{ij}}{\partial s_i}$ 的咽喉!

先看看 RankNet 的梯度

$$\frac{\partial L}{\partial w_k} = \sum_{(i,j)\in P} \frac{\partial L_{ij}}{\partial w_k} = \sum_{(i,j)\in P} \frac{\partial L_{ij}}{\partial s_i}\frac{\partial s_i}{\partial w_k} + \frac{\partial L_{ij}}{\partial s_j}\frac{\partial s_j}{\partial w_k},$$

注意有下面对称性

$$\begin{aligned}\frac{\partial L_{ij}}{\partial s_i} &= \frac{\partial\left\{\frac{1}{2}(1-S_{ij})\sigma(s_i-s_j) + \log\left\{1+\exp\big(-\sigma(s_i-s_j)\big)\right\}\right\}}{\partial s_i} \\ &= \frac{1}{2}(1-S_{ij})\sigma - \frac{\sigma}{1+\exp\big(\sigma(s_i-s_j)\big)} \\ &= -\frac{\partial L_{ij}}{\partial s_j}.\end{aligned}$$

于是我们定义

$$\lambda_{ij} \stackrel{\text{def}}{=} \frac{\partial L_{ij}}{\partial s_i} = -\frac{\partial L_{ij}}{\partial s_j},$$

考虑有序对 (i, j)，有 $S_{ij} = 1$，于是有简化

$$\lambda_{ij} \stackrel{\text{def}}{=} -\frac{\sigma}{1 + \exp\big(\sigma(s_i - s_j)\big)},$$

在此基础上，考虑评价指标 Z（比如 NDCG）的变化

$$\lambda_{ij} \stackrel{\text{def}}{=} -\frac{\sigma}{1 + \exp\big(\sigma(s_i - s_j)\big)} \cdot |\Delta Z_{ij}|.$$

对于具体的文档 x_i，有

$$\lambda_i = \sum_{(i,j)\in P} \lambda_{ij} - \sum_{(j,i)\in P} \lambda_{ij}.$$

每条文档移动的方向和趋势取决于其他所有与之 *label* 不同的文档.

我 (yuān) 们 (fāng) 做了什么?

- 分析梯度的物理意义;
- 绕开损失函数，直接定义梯度;
- 链式法则拆分原有损失函数的偏导，公式简洁速度加快.

反推 LambdaRank 的损失函数

$$L_{ij} = \log\Big\{1 + \exp\big(-\sigma(s_i - s_j)\big)\Big\} \cdot |\Delta Z_{ij}|.$$

4 LambdaMART

- MART 是一个框架，缺一个梯度;
- LambdaRank 直接定义了梯度.

元芳:「让它们在一起吧」.

Algorithm 1 LambdaMART 算法

procedure λ-MART($\{\vec{x}, y\}$, N, L, η)
  for $i : 0 \to \left|\{\vec{x}, y\}\right|$ **do** ▷ 初始化
    $F_0(\vec{x}_i) = \text{BaseModel}(\vec{x}_i)$ ▷ 如果没有 BaseModel，那么初始化为全 0
  end for
  for $k : 0 \to N$ **do** ▷ 迭代生成 N 棵树
    for $i : 0 \to \left|\{\vec{x}, y\}\right|$ **do** ▷ 遍历训练集
      $y_i = \lambda_i$ ▷ 计算每个文档的 λ-梯度
      $w_i = \dfrac{\partial y_i}{\partial F_{k-1}(\vec{x}_i)}$ ▷ 用于后续牛顿迭代法的偏导
    end for
    $\left\{R_{lk}\right\}_{l=1}^{L}$ ▷ 利用上述梯度，生成决策树，公式 ★
    for $l : 0 \to L$ **do**
      $\gamma_{lk} = \dfrac{\sum\limits_{\vec{x}_i \in R_{lk}} y_i}{\sum\limits_{\vec{x}_i \in R_{lk}} w_i}$ ▷ 牛顿迭代法，计算叶子节点的值
    end for
    for $i : 0 \to \left|\{\vec{x}, y\}\right|$ **do** ▷ 更新模型
      $F_k(\vec{x}_i) = F_{k-1}(\vec{x}_i) + \eta \sum_l \gamma_{lk} I(\vec{x}_i \in R_{lk})$ ▷ 根据学习率 η 更新每个文档的得分
    end for
  end for
end procedure