Learning to Rank 分享

介绍 LambdaMART 算法

jqian

2016-12-07

Rank 模型

传统 Rank 模型

Similarity-based model:

- Boolean model
- VSM

Probabilistic model:

- BM25

Hyperlink-based model:

- PageRank
- HITS

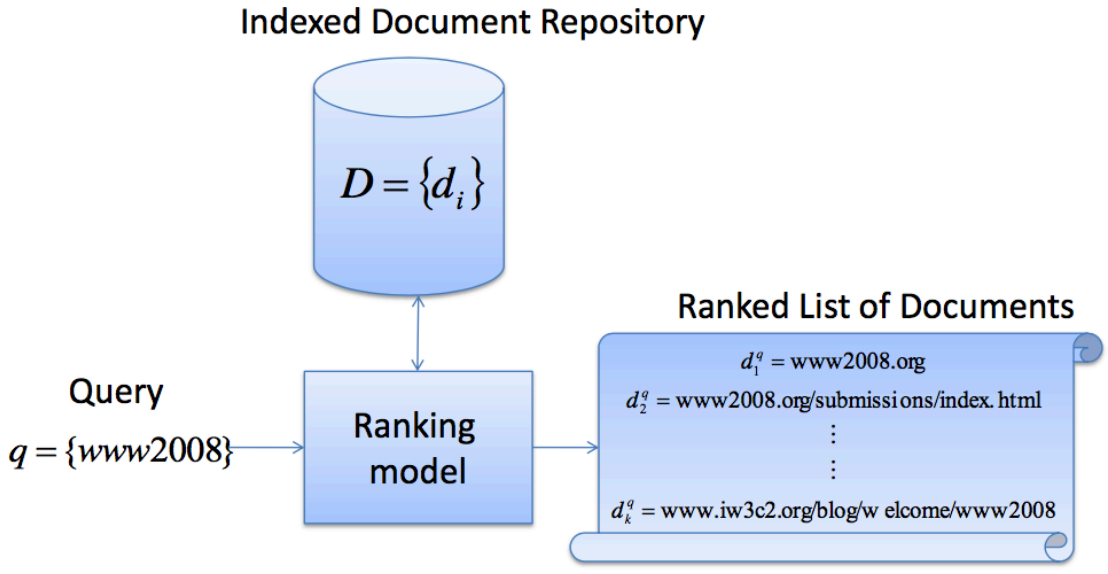

Figure 1：传统排序框架[1]

Tie-Yan Liu. LtR Tutorial @WWW2008

Lucene Scoring Function[2]

$$\begin{aligned}\text{score}(q,d) = &\text{queryNorm}(q) \\ &\cdot \text{coord}(q,d) \\ &\cdot \sum_{t \in q} \text{tf}(t \in d) \cdot \text{idf}(t)^2 \\ &\quad \cdot \text{boost}(t) \cdot \text{norm}(t,d)\end{aligned}$$

[2] https://www.elastic.co/guide/en/elasticsearch/guide/current/practical-scoring-function.html

传统 Rank 模型的缺陷

- 调参比较麻烦，尤其参数增多，容易过拟合
- 难以融合其他特征，也不太好做个性化

Rank 模型和 CTR 模型的区别

- Rank 模型关注顺序，文档具体 score 意义不大
- CTR 模型非常关注 ad ctr，因为和 ecpm 直接挂钩，且具备物理含义

Rank 指标

MAP (Mean Average Precision)

- Precision at position n

$$P@n = \frac{\#\{\text{relevant documents in top } n \text{ results}\}}{n}$$

- Average Precision

$$AP = \frac{\sum_n P@n \cdot I\{\text{document } n \text{ is relevant}\}}{\#\{\text{relevant documents}\}}$$

- MAP: averaged over all queries in the test set

MRR (Mean Reciprocal Rank)

- For query q_i, rank position of the first relevant document: r_i
- MRR: average of $1/r_i$ over all queries

NDCG (Normalized Discounted Cumulative Gain)

- NDCG at position n

$$N(n) = Z_n \sum_{j=1}^{n} \frac{\left(2^{r(j)} - 1\right)}{\log(1 + j)}$$

- NDCG averaged for all queries in test set

Rank 指标小结

- Query level
  - 在所有的 query 上取均值
- Position sensitive
  - 指标计算都是和位置相关的
- Non-smooth / non-differentiable
  - 基于文档的排序结果计算
  - 模型参数 $\rightarrow$ 文档 score $\rightarrow$ 指标

Learning to Rank 框架

ML 框架

- Input space X
- Output space Y
- Hypothesis space h
- Loss function l

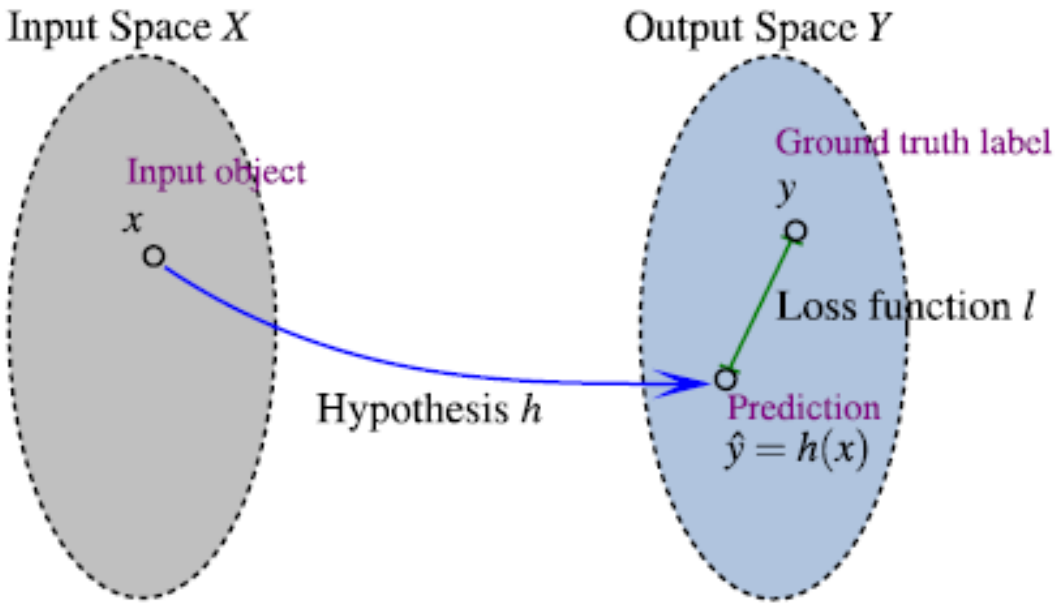

Figure 2: 机器学习框架

Learning to Rank 框架

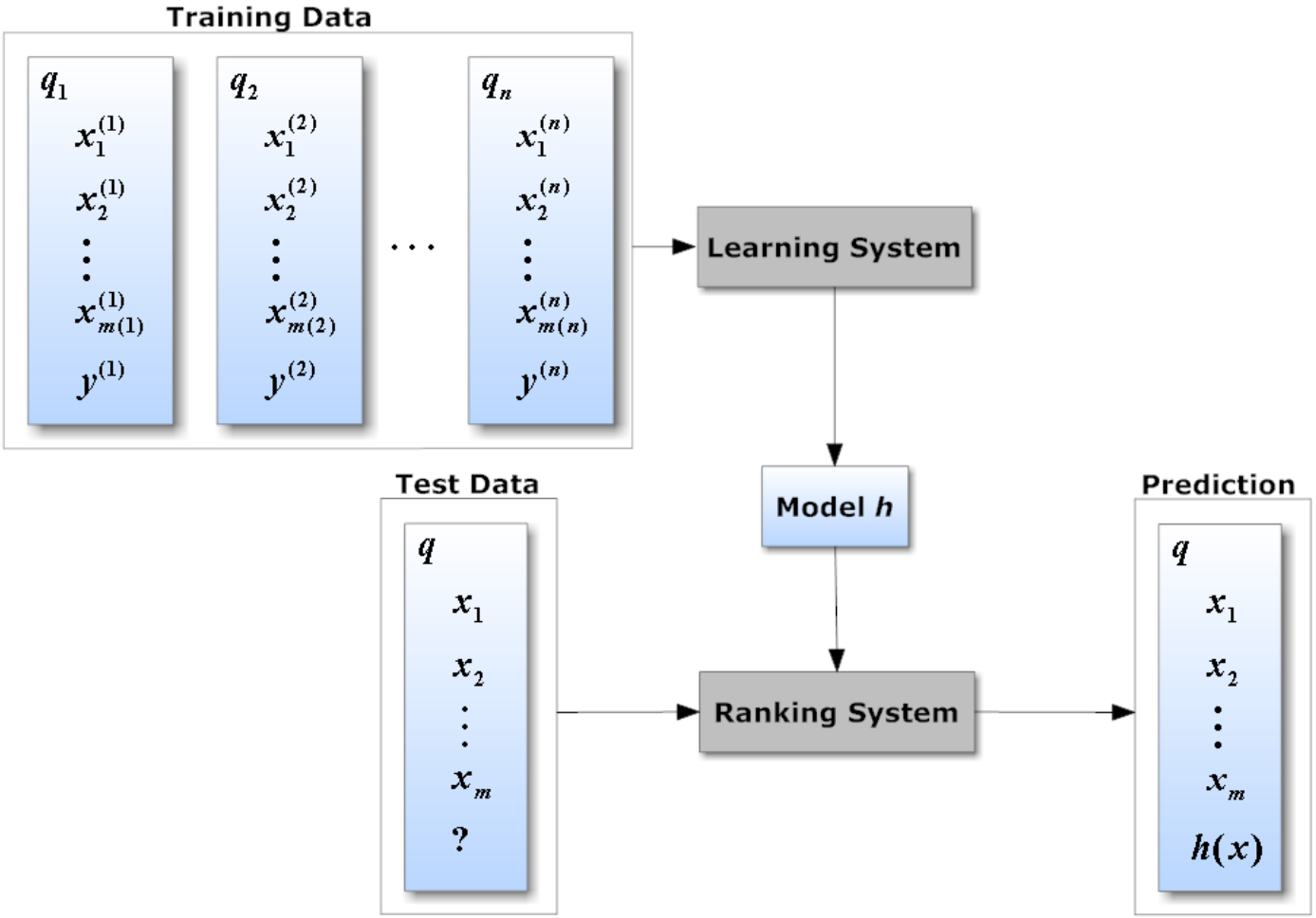

Figure 3: Learning to Rank 框架[3]

[3] Tie-Yan Liu. LtR Tutorial @WWW2009

Learning to Rank 算法

Pointwise

问题: regression、classification、ordinal regression

- X: 单个文档 x_i
- y: label 相关性
- h: 预估相关性
- L: RMSE、logloss

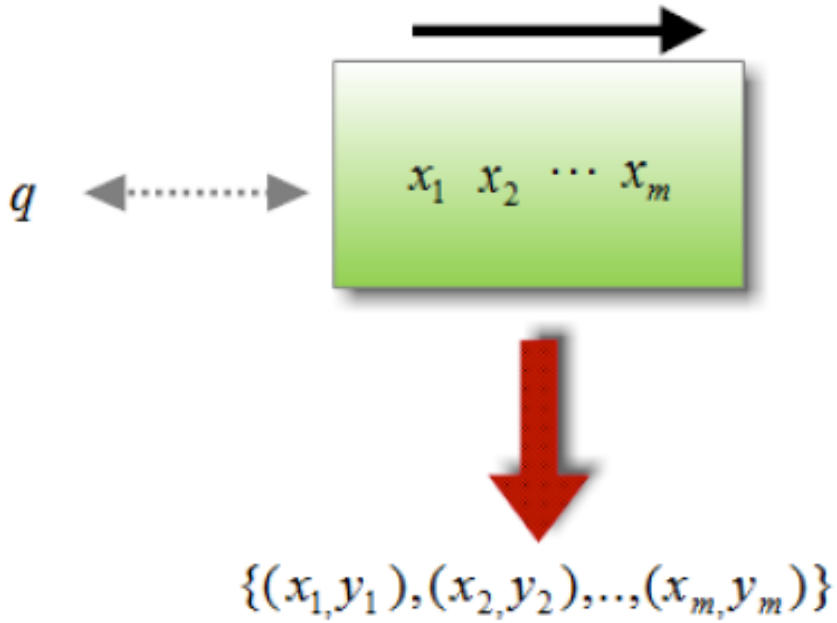

Figure 4: Pointwise 方法

Pointwise 方法

- Regression
  - GBDT
  - Additive Groves
- Classification
  - McRank 证明 DCG 的误差可以被分类误差给 bound 住
- Oridinal Regression
  - Prank 训练线性回归模型和各档位阈值

Pointwise 小结

- 假设相关度与 query 无关
  - 只要 (query, doc) 相关度相同，都被放到一个类别里
- 分类方法对头部相关类别区分不够
  - 同一类别的文档无法作出排序

示例

一个常见 query 和它的不相关 doc，它们之间的 tf 可能会比一个稀有 query 和它的相关 doc 之间的 tf 更高。这样就会导致训练数据不一致，难以取得好的效果。

Pairwise

二元分类问题

- X: 文档 pair (x_i, x_j)
- y: 每对 pair 的好坏
  $y_{i,j} \in \{+1, -1\}$
- h: 一般转化为二分类模型
- L: pairwise loss
  $L(h; x_i, x_j, y_{i,j})$

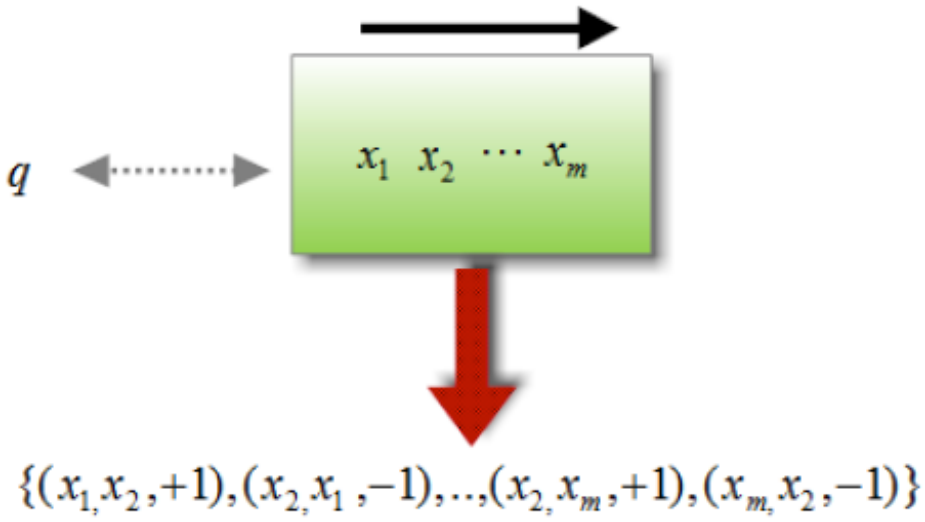

Figure 5: Pairwise 方法

Pairwise 方法

- Classification
  - SvmRank 利用 Kendall 相关系数度量两类 pair，使用 SVM 优化该目标
  - GBRank 利用 boosting 思想，每次利用分错的 pair 样本来训练模型
  - RankNet 两层神经网络，优化 pair 顺序概率的交叉熵损失
  - FRank 类似 RankNet，使用 Fidelity 损失函数

Pairwise 小结

- 不同 label 之间的区分度一致对待
  - 更倾向于头部点击，所以对相关对高的文档应该更好的区分
- 相关文档集合大小带来的 model bias
  - 对相关文档集合小的 query 产生的训练不足
- 不是直接以 rank 指标为优化目标
  - 模型本身的 bias
  - 容易过拟合

示例

Ideal: p g g b b b b

ranking 1: g p g b b b b : one wrong pair Worse

ranking 2: p g b g b b b : one wrong pair Better

Listwise

- X: 文档集合 $\mathbf{x} = \{x_i\}_{i=1}^{m}$
- y: 有序排列 $\mathbf{y} = \{y_i\}_{i=1}^{m}$
- h: 一般拆解为 sort $\cdot f(\mathbf{x})$
- L: 逆序数量

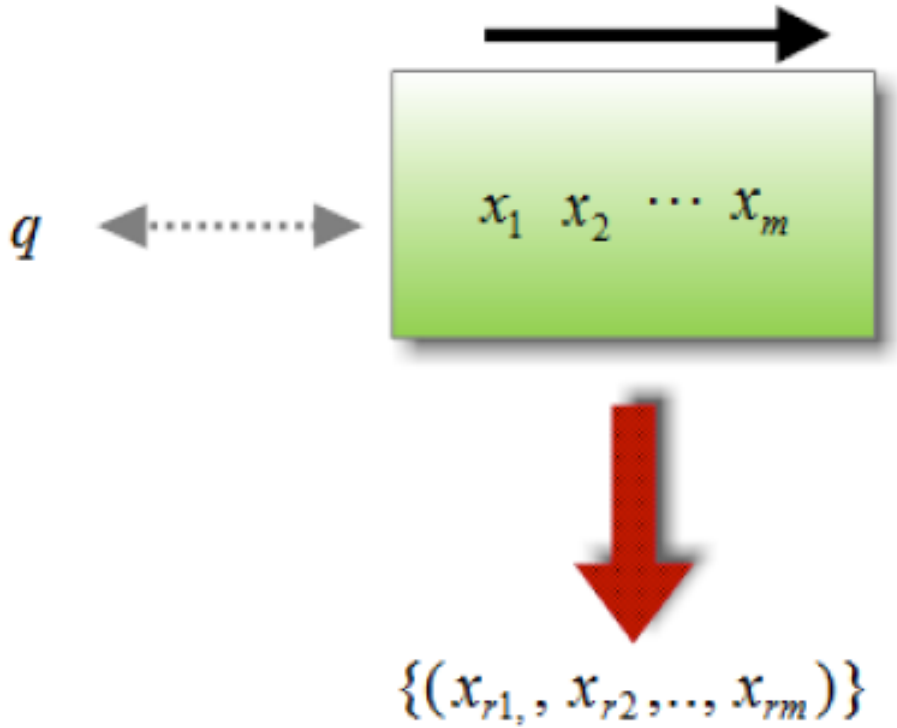

Figure 6: Listwise 方法

Listwise 方法

- 直接优化 IR 指标
  - SoftRank 计算 SoftNDCG 替代 NDCG
  - LambdaRank/LambdaMART 直接定义梯度优化 IR 指标
- 定义 listwise loss
  - RankCosine 使用文档得分向量的 cos 相似度作为 loss 函数
  - ListNet 使用排列的概率分布之间的 KL 散度作为 loss 函数
  - ListMLE 使用排列的 log likelihood 作为 loss 函数

Listwise 小结

- LambdaMART 获得 Yahoo! 2010 年 LtR 比赛第一
- 在微软和 Yahoo! 搜索上应用较多
- 国内淘宝、京东等排序和个性化推荐场景均有应用

问题

- SoftNDCG 不能完全替代 NDCG
- Lambda 梯度隐含的 loss 函数和 NDCG 关系并无理论证明
- 多个 IR 指标之间并非保持一致

LambdaMART 算法

符号说明

Table 1: 符号说明

符号	说明
x_i	文档 i 的特征向量
s_i	文档 i 的 score
$f(x_i; w)$	文档 i 的 score function（model）
U_i	文档 i 的 URL（文档标识）
$\{i, j\}$	文档 i 和 j 组成的 pair
P_{ij}	文档 i 排在 j 之前的预测概率
$\bar{P}_{ij}$	文档 i 排在 j 之前的实际概率
$U_i \rhd U_j$	文档 i 和 j 的偏序关系
$S_{i,j} \in \{0, \pm 1\}$	文档 i 和 j 的偏序关系值
I	$\{i, j\}$ 集合，且 $U_i \rhd U_j$

RankNet

目标是学习 $f(x_i; w)$，RankNet 忽略了 NDCG 指标，只考虑 pairwse 顺序。

实际概率：$\bar{P}_{ij} \stackrel{\text{def}}{=} \frac{1}{2}(1 - S_{ij})$

预测概率：$P_{ij} = P(U_i \rhd U_j) \stackrel{\text{def}}{=} \frac{1}{1+e^{-\sigma(s_i - s_j)}}$ [4]

交叉熵损失

$$
\begin{aligned}
C_{ij} &= -\bar{P}_{ij} \log P_{ij} - (1 - \bar{P}_{ij}) \log(1 - P_{ij}) \\
&= \frac{1}{2}(1 - S_{ij})\sigma(s_i - s_j) + \log\left(1 + e^{-\sigma(s_i - s_j)}\right)
\end{aligned}
$$

[4] σ 用来调整 sigmoid 函数形状

求解梯度

用 SGD 求解模型参数 w

$$w_k \rightarrow w_k - \eta \frac{\partial C}{\partial w_k} = w_k - \eta \left(\frac{\partial C}{\partial s_i} \frac{\partial s_i}{\partial w_k} + \frac{\partial C}{\partial s_j} \frac{\partial s_j}{\partial w_k} \right)$$

求解出 s_i 和 s_j 的偏导数，具有对称性

$$\frac{\partial C}{\partial s_i} = \sigma \left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + e^{\sigma(s_i - s_j)}} \right) = -\frac{\partial C}{\partial s_j}$$

RankNet 算法

具体打分函数 $f(x; w)$ 只要是光滑可导就行，比如 $f(x) = wx$ 都可以，RankNet 使用了两层神经网络

$$f(x) = g^3\left(\sum_j w_{ij}^{32} g^2\left(\sum_k w_{jk}^{21} x_k + b_j^2\right) + b_i^3\right)$$

问题

每学习一对 pair 就要更新一次参数，对 BP 网络太慢了。

加速 RankNet

继续分解梯度式子

$$
\begin{aligned}
\frac{\partial C}{\partial w_k} &= \frac{\partial C}{\partial s_i}\frac{\partial s_i}{\partial w_k} + \frac{\partial C}{\partial s_j}\frac{\partial s_j}{\partial w_k} \\
&= \sigma\left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + e^{\sigma(s_i - s_j)}}\right)\left(\frac{\partial s_i}{\partial w_k} - \frac{\partial s_j}{\partial w_k}\right) \\
&= \lambda_{ij}\left(\frac{\partial s_i}{\partial w_k} - \frac{\partial s_j}{\partial w_k}\right)
\end{aligned}
$$

这里定义：

$$
\lambda_{ij} \stackrel{\text{def}}{=} \sigma\left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + e^{\sigma(s_i - s_j)}}\right)
$$

一次梯度下降量 δw

用 λ_{ij} 重新表示 $\frac{\partial C}{\partial w_k}$，得到参数 w_k 的更新之和

$$\delta w_k = -\eta \sum_{\{i,j\}\in I} \left(\lambda_{ij} \frac{\partial s_i}{\partial w_k} - \lambda_{ij} \frac{\partial s_j}{\partial w_k} \right) \stackrel{\text{def}}{=} -\eta \sum_i \lambda_i \frac{\partial s_i}{\partial w_k}$$

这里用到集合 I 的对称性

$$\sum_{\{i,j\}\in I} \lambda_{ij} \frac{\partial s_j}{\partial w_k} = \sum_{\{j,i\}\in I} \lambda_{ij} \frac{\partial s_i}{\partial w_k}$$

mini-batch 和 λ_i

现在，一次参数更新只和 U_i 有关

$$\lambda_i = \sum_{\{i,j\} \in I} \lambda_{ij} - \sum_{\{j,i\} \in I} \lambda_{ij}$$

λ_i 可以理解为文档 i 在 query 排列中移动的方向和力度。

示例

如果只有一组 pair $U_1 \rhd U_2$，则 $I = \{\{1, 2\}\}$。
可知 $\lambda_1 = \lambda_{12} = -\lambda_2$。

mini-batch

加和关于 U_i 的所有 pair 的贡献后，才更新一次 w，极大的加快 RankNet 训练过程。

RankNet 梯度分析

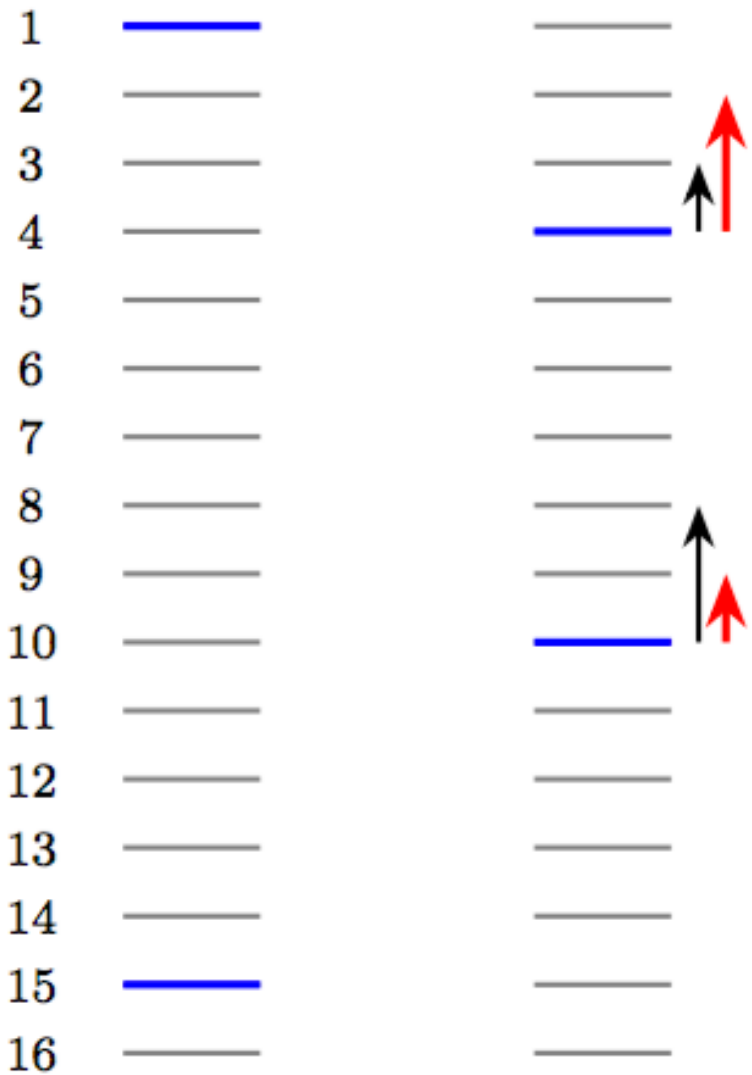

Figure 7: 梯度分析示意图

- RankNet 的梯度下降表现在结果整体变化中，是逆序对减少 ($13 \rightarrow 11$);
- RankNet 的梯度下降表现在单个的变化中，是文档在列表中的移动趋势 (黑色箭头);
- NDCG 关注头部排序情况，理想情况下文档的移动趋势如红色箭头所示。

LambdaRank 尝试直接定义这种移动趋势。

Lambda 梯度

LambdaRank 在 RankNet 梯度基础上，考虑评价指标 Z 的变化[5]，拟定效用函数 C 的梯度

$$\lambda_{ij} = \frac{\partial C(s_i - s_j)}{\partial s_i} \stackrel{\text{def}}{=} -\frac{\sigma}{1 + e^{\sigma(s_i - s_j)}} \cdot |\Delta Z_{ij}|$$

对于具体的文档 x_i，容易得到所谓的 **Lambda 梯度**

$$\lambda_i = \sum_{\{i,j\} \in I} \lambda_{ij} - \sum_{\{j,i\} \in I} \lambda_{ij}.$$

每篇文档的移动趋势取决于其他相关度 label 不同的文档。

[5] 只需要关注 $S_{ij} = 1$ 的情况

MART

MART 目标是寻找使得期望损失最小的决策函数

$$F^* = \arg\min_F E_{y,\mathbf{x}}\left(L(y, F(\mathbf{x}))\right)$$

这个决策函数具备一定形式，即一组弱学习器的加性组合

$$F(\mathbf{x}; \rho_m, \mathbf{a}_m) = \sum_{m=0}^{M} \rho_m h(\mathbf{x}; \mathbf{a}_m)$$

MART 采用贪心策略，每步迭代的目标均最小化 loss，即减少

$$\tilde{y}_i = \left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$$

使用 logloss 的 MART

Algorithm 5: L_2_TreeBoost

$F_0(\mathbf{x}) = \frac{1}{2}\log\frac{1+\bar{y}}{1-\bar{y}}$

For $m = 1$ to M do:

$\tilde{y}_i = 2y_i / (1 + \exp(2y_i F_{m-1}(\mathbf{x}_i))),\ i = 1, N$

$\{R_{jm}\}_1^J = J$–terminal node $tree(\{\tilde{y}_i, \mathbf{x}_i\}_1^N)$

$\gamma_{jm} = \sum_{\mathbf{x}_i \in R_{jm}} \tilde{y}_i / \sum_{\mathbf{x}_i \in R_{jm}} |\tilde{y}_i| (2 - |\tilde{y}_i|),\ j = 1, J$

$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \sum_{j=1}^J \gamma_{jm} 1(\mathbf{x} \in R_{jm})$

endFor

end Algorithm

Figure 8: 使用 logloss 的 GBDT 算法

LambdaMART

效用函数

$$\sum_{\{i,j\}\rightleftharpoons I} \lambda_{ij} \stackrel{\text{def}}{=} \sum_{\{i,j\}\in I} \lambda_{ij} - \sum_{\{j,i\}\in I} \lambda_{ij}$$

对文档 i 可以反推出效用函数（utility function）

$$C = \sum_{\{i,j\}\rightleftharpoons I} |\Delta Z_{ij}| \log\left(1 + e^{-\sigma(s_i - s_j)}\right)$$

C 的二次偏导

$$w_i = \frac{\partial^2 C}{\partial s_i^2} = \sum_{\{i,j\}\rightleftharpoons I} \sigma^2 |\Delta Z_{ij}| \rho_{ij}(1 - \rho_{ij})$$

其中，$\rho_{ij} \stackrel{\text{def}}{=} \frac{1}{1+e^{\sigma(s_i - s_j)}} = \frac{-\lambda_{ij}}{\sigma|Z_{ij}|}$

LambdaMART 算法

Algorithm 1 The LambdaSMART algorithm.

for $i = 0$ to N **do**
  $F_0(x_i) = BaseModel(x_i)$ \\$BaseModel$ may be empty or set to a submodel.
end for
for $m = 1$ to M **do**
  for $i = 0$ to N **do**
    $y_i = \lambda_i$ \\Calculate λ-gradient for sample i.
    $w_i = \frac{\partial y_i}{\partial F(x_i)}$ \\Calculate derivative of λ-gradient for sample i.
  end for
  $\{R_{lm}\}_{l=1}^{L}$ \\Create L-terminal node tree on $\{y_i, x_i\}_{i=1}^{N}$.
  for $l = 0$ to L **do**
    $\gamma_{lm} = \frac{\sum_{x_i \in R_{lm}} y_i}{\sum_{x_i \in R_{lm}} w_i}$ \\Find the leaf values based on approximate Newton step.
  end for
  for $i = 0$ to N **do**
    $F_m(x_i) = F_{m-1}(x_i) + v \sum_l \gamma_{lm} 1(x_i \in R_{lm})$ \\Update model based on approximate Newton step and shrinkage size.
  end for
end for

Figure 9: LambdaMART 算法

LambdaMART 的优点

- 直接求解排序问题，而不是 pointwise 或 pairwise 方法；
- 直接定义了 IR 指标的梯度，具有明确的物理意义；
- 可以在已有模型的基础上进行持续训练；
- 每次节点分裂选取 gain 最大的特征，自带特征选择；
- 对正负例的数量比例不敏感。

LambdaMART 实现

Ranklib.learn()

```
1 for 1..nTrees:
2     computePseudoResponses();
3     hist.update(pseudoResponses);  // 更新直方图
4     RegressionTree rt ... fit() ;  // 训练单棵树
5     ensemble.add(rt, learningRate);
6     updateTreeOutput(rt);          // 更新回归树的输出
7     for(int i=0;i<modelScores.length;i++)  // 更新score
8         modelScores[i] += learningRate *
              rt.eval(martSamples[i]);
9     computeModelScoreOnTraining(); // 计算指标
```

LambdaMART.computePseudoResponses()

```
1  // 获取一次Query请求的文档
2  RankList r = samples.get(i);
3  // 计算交换顺序后的NDCG@K
4  float[][] changes = computeMetricChange(i, current);
5  for(int j=0;j<r.size();j++) {
6      DataPoint p1 = r.get(j);
7      for(int k=0;k<r.size();k++) {
8          DataPoint p2 = r.get(k);
9          double deltaNDCG = Math.abs(changes[j][k]);
10         if(p1.getLabel() > p2.getLabel()) {
```

LambdaMART.computePseudoResponses()

```
11                double rho = 1.0 / (1 +
                      Math.exp(modelScores[current+j] -
                      modelScores[current+k]));
12                // lambda梯度，作为训练回归树的label
13                double lambda = rho * deltaNDCG;
14                lambdas[j] += lambda; lambdas[k] -= lambda;
15                // delta是lambda的偏导数，step size
16                double delta = rho * (1.0 - rho) * deltaNDCG;
17                // 用于后续计算gamma值
18                weights[j] += delta; weights[k] += delta;
19            }
20        }
21    }
```

LambdaMART.updateTreeOutput(RegressionTree)

```
1 Split s = leaves.get(i);
2 int[] idx = s.getSamples();
3 for(int j=0;j<idx.length;j++) {
4     int k = idx[j];
5     s1 += pseudoResponses[k];   // y_i
6     s2 += martSamples[k].getCached(); // w_i
7 }
8 s.setOutput(s1/s2);   // gamma
```

对比 MART pointwise 实现

computePseudoResponses()

直接使用样本的实际 label 和预测 label 的残差。

```
for(int i=0;i<martSamples.length;i++)
    pseudoResponses[i] = martSamples[i].getLabel() -
        modelScores[i];
```

updateTreeOutput(RegressionTree rt)

直接输出 residual 均值。

总结

LtR 数据集

MSLR-WEB10K/30K

来自 Bing 的搜索数据[a]，label 分 5 档：

0 qid:1 1:3 2:0 3:2 4:2 ... 135:0 136:0
2 qid:1 1:3 2:3 3:0 4:0 ... 135:0 136:0

[a] https://www.microsoft.com/en-us/research/project/mslr/

Feature List

136 维特征：

- Statistic, min/max/sum/mean/variance
- TF-IDF
- BM25 / VSM
- PageRank / SiteRank

业界的应用

搜索

- Microsoft 和 Yahoo! 应用比较多
- Google 搜索相对广告 ML 应用较少

推荐

- 应用较广，因为大多数推荐问题都以 Ranked List 提供结果
- 基于 LR/GBDT pointwise 模型较多

工具

- Ranklib
  https://people.cs.umass.edu/~vdang/ranklib.html
- Xgboost https://github.com/dmlc/xgboost
- LightGBM https://github.com/Microsoft/LightGBM

Reference

1. Chris Burges. *From RankNet to LambdaRank to LambdaMART: An Overview*. MSR-TR-2010-82.
2. Zhe Cao. *Learning to Rank: From Pairwise Approach to Listwise Approach*. ICML 2007.
3. J.H. Friedman. *Greedy Function Approximation: A Gradient Boosting Machine*. IMS 2001.
4. Tie-Yan Liu. *Learning to Rank for Information Retrieval*. Tutorial at WWW 2008.