Listwise Learning to Rank

Research Preparation Exam for

Harsh Pareek (hhp266)

harshp@cs.utexas.edu
Outline

• Learning to Rank
  – Probability Ranking Principle
  – Datasets and Evaluation Metrics

• Loss Functions for Learning to Rank
  – Pointwise, Pairwise and Listwise loss functions
  – LambdaMART

• Listwise Ranking Functions [Pareek and Ravikumar 2014]
  – Need for Listwise Ranking Functions
  – A Representation Theorem
Ranking

- Central problem in Information Retrieval
  - Applies to many problems: document retrieval, recommender systems (collaborative filtering), product rating etc.

- Ranking Problems:
  - Rank the documents purely according to their relevance with regards to the query.
  - Consider relations between documents such as similarity, website structure, and diversity
  - Aggregate several candidate ranked lists to get a better ranked list. Called Rank Aggregation or meta search
Learning to Rank (LETOR)

- Machine Learning-based approach to Ranking
- **Probability Ranking Principle:** For each item to be ranked, predict a score, often a probability or an expected value. e.g.,
  - $P(\text{user will click this link})$
  - $P(\text{user will purchase this item})$
  - $E[\text{Revenue earned by showing this item}]$
- Learn weights on features to get a **scoring function**
- Use a loss function to minimize some appropriate **Evaluation Metric**
Yahoo! LETOR dataset

- Queries were randomly sampled from Yahoo! Search query logs
- For each query, documents collected using various base IR systems
- Document relevance scored by expert judges on a scale of 0 (bad) to 4 (perfect)
- Document features:
  - Web Graph features: PageRank, number of inlinks, outlinks
  - Document statistics: Word count features
  - Document classifier: Topic, quality, type of page, spam, adult
  - Query-document compatibility: Text match features
  - Click features: Probability of click/skip, dwell time
  - Page Freshness
# Datasets for Learning to Rank

Chapelle et al (2010)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Queries</th>
<th>Docs.</th>
<th>Rel.</th>
<th>Features</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>LETOR 3.0 – Gov</td>
<td>575</td>
<td>568 k</td>
<td>2</td>
<td>64</td>
<td>2008</td>
</tr>
<tr>
<td>LETOR 3.0 – Ohsumed</td>
<td>106</td>
<td>16 k</td>
<td>3</td>
<td>45</td>
<td>2008</td>
</tr>
<tr>
<td>LETOR 4.0</td>
<td>2,476</td>
<td>85 k</td>
<td>3</td>
<td>46</td>
<td>2009</td>
</tr>
<tr>
<td>Yandex</td>
<td>20,267</td>
<td>213 k</td>
<td>5</td>
<td>245</td>
<td>2009</td>
</tr>
<tr>
<td>Yahoo!</td>
<td>36,251</td>
<td>883 k</td>
<td>5</td>
<td>700</td>
<td>2010</td>
</tr>
<tr>
<td>Microsoft</td>
<td>31,531</td>
<td>3,771 k</td>
<td>5</td>
<td>136</td>
<td>2010</td>
</tr>
</tbody>
</table>
Evaluation Metrics for LETOR

• Used to compare the output of a ranking algorithm to ground truth data

• Represents a user model
  – Penalize errors in later positions more
    • E.g. An error in the 101st position should be penalized less than an error in the 3rd position
  – Account for Diversity
    • Penalize algorithms that place too many similar items at the top of the ranking
Evaluation Metrics: MAP

Mean Average Precision

\[ P@k(q) = \frac{\#{\text{relevant documents in top-}k\text{ positions}}}{k} \]

Average Precision

\[ AP(q) = \sum_{k=1}^{m} \frac{P@k(q) \cdot l_k}{\#{\text{relevant docs}}} \]
Evaluation Metrics: NDCG

- Normalized Discounted Cumulative Gain

\[
DCG@k = \sum_{i=1}^{k} \frac{2^{l_i} - 1}{\log_2(1 + i)}
\]

\[
NDCG@k = \frac{DCG}{Ideal\ DCG}
\]
Evaluation Metrics: ERR

• Inspired by a Cascade User Model

\[
\text{ERR} = \sum_{i=1}^{n} \frac{1}{i} P(\text{user stops at } i)
\]

\[
= \sum_{\{i=1\}}^{n} \frac{1}{i} R(y_i) \prod_{j=1}^{i-1} (1 - R(y_j))
\]

− Where \( R(y) = \frac{2^y - 1}{16} \)
Evaluation Metrics: Diversity

• Extrinsic Diversity:
  – Hedge against uncertainty about the user’s information need
  – E.g. A web search query for “Jaguar” could return the car, the animal, the sports team etc.

• Intrinsic Diversity:
  – Avoid redundancy
  – E.g. In the Facebook News Feed, show only one news article about the Syrian Refugee Crisis
Loss functions for Learning to Rank

- Loss functions are be categorized as:
  - Pointwise: Estimate relevance for each individual document. Similar to regression.
  - Pairwise: Ignore individual document relevances. Instead, relative rankings should be correct.
  - Listwise: Optimize IR metrics directly. Can take into account position and diversity
Pointwise loss function

- Cast the problem as a regression, classification or Ordinal Regression problem, e.g.
  - Subset ranking using Regression [Cossock and Zhang, 2006] uses the square loss
    \[ L(f; x_j, y_j) = (y_j - f(x_j))^2. \]
  - McRank[Li et al, 2007]: Adapts multiclass classification techniques for ranking
Pairwise Loss Functions

For each pair of documents, $x_u$ and $x_v$, minimize the loss,

$$L(h; x_u, x_v, y_{u,v}) = \frac{|y_{u,v} - h(x_u, x_v)|}{2},$$

To predict a ranking at test time, perform Rank Aggregation on the partial rankings for each pair.

$$\max_{\pi} \sum_{u < v} h(x_{\pi^{-1}(u)}, x_{\pi^{-1}(v)}).$$
Pairwise Loss Functions

SVMRank: Using \( f(x) = w^T x \),

\[
\begin{align*}
\min & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \sum_{u,v:y_u,v=1} \xi_{u,v}^{(i)} \\
\text{s.t.} & \quad w^T(x_u^{(i)} - x_v^{(i)}) \geq 1 - \xi_{u,v}^{(i)}, \quad \text{if } y_u^{(i)} = 1, \\
& \quad \xi_{u,v}^{(i)} \geq 0, \quad i = 1, \ldots, n.
\end{align*}
\]
Listwise Losses: AdaRank

For $M(f, x, y)$, an IR measure of interest

**Algorithm 2 Learning Algorithms for AdaRank**

**Input:** document group for each query  
**Given:** initial distribution $D_1$ on input queries  
**For** $t = 1, \ldots, T$

- Train weak ranker $f_t(\cdot)$ based on distribution $D_t$.  
- Choose $\alpha_t = \frac{1}{2} \log \frac{\sum_{i=1}^{n} D_t(i)(1+M(f_t, x^{(i)}, y^{(i)}))}{\sum_{i=1}^{n} D_t(i)(1-M(f_t, x^{(i)}, y^{(i)}))}$  
- Update $D_{t+1}(i) = \frac{\exp(-M(\sum_{s=1}^{t} \alpha_s f_s, x^{(i)}, y^{(i)}))}{\sum_{j=1}^{n} \exp(-M(\sum_{s=1}^{t} \alpha_s f_s, x^{(j)}, y^{(j)}))}$  

**Output:** $\sum_{t} \alpha_t f_t(\cdot)$. 

Listwise Losses

• ListNet: Based on the Luce Model for Rankings
  – Minimizes the K-L divergence between the labeling predicted by the algorithm and the permutation implied by relevance labels

• [Ravikumar et al 2011] provide an exhaustive characterization of listwise surrogates for NDCG
  – ListNet is a special case for the Cross-entropy Loss
Listwise Losses

$\textit{SVM}^{\text{MAP}}$ : Optimizes the MAP directly

$$\min \frac{1}{2} \|w\|^2 + \frac{C}{n} \sum_{i=1}^{n} \xi^{(i)}$$

s.t. $\forall y^{c(i)} \neq y^{(i)}$, 

$$w^T \Psi(y^{(i)}, x^{(i)}) \geq w^T \Psi(y^{c(i)}, x^{(i)}) + 1 - \text{AP}(y^{c(i)}) - \xi^{(i)}.$$ 

$$\Psi(y, x) = \sum_{u,v:y_u=1,y_v=0} (x_u - x_v).$$

$$\Psi(y^{c}, x) = \sum_{u,v:y_u=1,y_v=0} (y_u^{c} - y_v^{c})(x_u - x_v).$$
Yahoo! LETOR Results

• The challenge was dominated by methods based on Gradient Boosted Decision Trees
  – LambdaMART won the competition, though many other methods were as competitive

• Pointwise, Pairwise and Listwise methods performed almost equally well
  – With large amounts of data, loss functions are less important than the ranking/scoring functions used
LambdaMART [Burges, 2010]

Listwise loss trained using Gradient Boosted Decision Trees

LambdaMART

Algorithm: LambdaMART

set number of trees $N$, number of training samples $m$, number of leaves per tree $L$, learning rate $\eta$

for $i = 0$ to $m$ do

$F_0(x_i) = \text{BaseModel}(x_i)$  //If BaseModel is empty, set $F_0(x_i) = 0$

end for

for $k = 1$ to $N$ do

for $i = 0$ to $m$ do

$y_i = \lambda_i$

$w_i = \frac{\partial y_i}{\partial F_{k-1}(x_i)}$

end for

$\{R_{lk}\}_{l=1}^L$  // Create $L$ leaf tree on $\{x_i, y_i\}_{i=1}^m$

$\gamma_k = \frac{\sum_{x_i \in R_{lk}} y_i}{\sum_{x_i \in R_{lk}} w_i}$  // Assign leaf values based on Newton step.

$F_k(x_i) = F_{k-1}(x_i) + \eta \sum_l \gamma_k I(x_i \in R_{lk})$  // Take step with learning rate $\eta$.

end for

$\lambda_{ij} \equiv S_{ij} \left| \Delta \text{NDCG} \frac{\partial C_{ij}}{\partial o_{ij}} \right|$

$\lambda_i = \sum_{j \in P} \lambda_{ij}.$
Our contributions

LISTWISE RANKING FUNCTIONS
A Gap in the Literature

• Loss functions vs Ranking functions
  – Most work in the literature uses pointwise scoring/ranking functions with pointwise, pairwise or listwise loss functions

• This is insufficient for listwise metrics such as diversity
  – And is insufficient for NDCG/ERR as well!
Listwise Ranking Functions

• Pointwise Ranking Function
  – Scores each document individually at test time
  – Sort scores to obtain final ranking

• Pairwise Ranking Function
  – Each pairs of documents is scored. Pairs are aggregated to obtain a ranking

• Listwise Ranking Function
  – The score for a document depends on the other documents in the retrieved set
Rethinking the Probability Ranking Principle

• What if $P(\text{click})$ depends on the order of documents?
  $- P(\text{click}_1, \text{click}_2) \neq P(\text{click}_2, \text{click}_1)$

• What if $P(\text{click})$ for a document depends on the documents that have come before it
  $- P(\text{click}_1, \text{click}_2) \neq P(\text{click}_1)P(\text{click}_2)$
Listwise Ranking Functions

• Exchangeable LRF: For some $g$ (indep of $i$),
  $$f(x)_i = g(x_i, \{x_{\setminus i}\})$$
  – $g$ is symmetric in $\{x_{\setminus i}\}$

• Every pointwise ranking functions is a listwise ranking function, but the converse is not true
A Representation Theory for Listwise Ranking Functions

- An exchangeable ranking function can be decomposed, for some $g_t$, as:

$$f(x)_i = h(x_i, \{x_{\setminus i}\}) = \prod_{j \neq i} g_t(x_i, x_j)$$

- Thus, pairwise $g_t$ can be combined to obtain a listwise ranking function
Proofs for Representation Theorem

- Proofs of the representation theorem:
  - Tensor decomposition: Extension of Eigenvector factorization for a matrix
  - Functional Analysis: Similar to Eigenfunction factorization for a linear operator
  - De Finetti’s theorem: There exists $\theta$ such that
    \[ p(x_1, x_2, x_3, \ldots) = \int \prod_i p(x_i | \theta) d\theta \]
Listwise Ranking Functions in the Literature

Listwise ranking functions have been previously proposed in the literature by directly minimizing listwise losses.

• Qin et al. [2008] propose the following functions, obtained by optimizing continuous CRFs in closed form:

\[
\begin{align*}
\mathbf{f}(\mathbf{X}) &= (\mathbf{\alpha}^T \mathbf{e} \mathbf{I} + \beta (\mathbf{D} - \mathbf{S}))^{-1} \mathbf{X} \mathbf{\alpha} \\
\mathbf{f}(\mathbf{X}) &= \frac{1}{\mathbf{\alpha}^T \mathbf{e}} (2 \mathbf{X} \mathbf{\alpha} + \beta (\mathbf{D}_r - \mathbf{D}_c) \mathbf{e})
\end{align*}
\]

• Weston and Blitzer [2012] propose optimizing the following listwise loss directly at inference time (Latent Structured Ranking):

\[
\begin{align*}
\mathbf{f} (\mathbf{q}) &= \arg \max_d \sum_{i=1}^{m} \mathbf{w}_i \mathbf{q}^T \mathbf{U}^T \mathbf{V} \mathbf{d}_i + \sum_{i,j=1}^{m} \mathbf{w}_i \mathbf{w}_j \mathbf{d}_i^T \mathbf{S}^T \mathbf{S} \mathbf{d}_j
\end{align*}
\]
Using the Representation Theory

• Consider for some base ranker $b$ and similarity kernels $S_k$:

\[
  f_i(x) = b(x_i) \prod_{j \neq i} g(x_i, x_j; w)
  = b(x_i) \prod_{j \neq i} \exp \sum_k w_k S_k(x_i, x_j)
\]

  – With preprocessing, computational complexity at training time equivalent to that for pointwise ranking functions.
Experimental results [Pareek and Ravikumar 2014]

Table 1: Results for our reranking procedure across LETOR 3.0 datasets. For each dataset, the first column is the base ranker, second column is the loss function used for reranking.

<table>
<thead>
<tr>
<th></th>
<th>OHSUMED</th>
<th>TD2003</th>
<th>NP2003</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Base</td>
<td>Reranked w/ Cross Ent</td>
<td>Base</td>
</tr>
<tr>
<td></td>
<td>RankBoost</td>
<td></td>
<td>CA</td>
</tr>
<tr>
<td>ndcg@1</td>
<td>0.5104</td>
<td>0.5421</td>
<td>0.3500</td>
</tr>
<tr>
<td>ndcg@2</td>
<td>0.4798</td>
<td>0.4901</td>
<td>0.2875</td>
</tr>
<tr>
<td>ndcg@5</td>
<td>0.4547</td>
<td>0.4615</td>
<td>0.3228</td>
</tr>
<tr>
<td>ndcg@10</td>
<td>0.4356</td>
<td>0.4445</td>
<td>0.3210</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>HP2003</th>
<th>HP2004</th>
<th>NP2004</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Base</td>
<td>Reranked w/ Cross Ent</td>
<td>Base</td>
</tr>
<tr>
<td>ndcg@1</td>
<td>0.6667</td>
<td>0.7333</td>
<td>0.5200</td>
</tr>
<tr>
<td>ndcg@2</td>
<td>0.7667</td>
<td>0.7667</td>
<td>0.6067</td>
</tr>
<tr>
<td>ndcg@5</td>
<td>0.7546</td>
<td>0.7618</td>
<td>0.7034</td>
</tr>
<tr>
<td>ndcg@10</td>
<td>0.7740</td>
<td>0.7747</td>
<td>0.7387</td>
</tr>
</tbody>
</table>
Challenges in using Listwise Ranking Functions

• Test time evaluation cost:
  – LRFs may require $O(m^2)$ computations at test time
  – Compared to greedy approaches for diversification, this algorithm may be more efficient and more parallelizable

• Difficulty of collecting Training Data:
  – Accounting for Presentation Bias
  – Off-policy evaluation leads to biased training data
SUMMARY
Learning to Rank

Credit: Liu et al (2009)
Summary

• Considerations in Learning to Rank
  – Evaluation Metric
  – Loss Function
  – Ranking Function

• Listwise Ranking Functions
  – A Representation Theorem
Thank you!

Questions?
References

• The material in this presentation is inspired by
  – Liu et al. 2009
  – Chapelle et al. 2011