BoltzRank: Learning to Rank by Maximizing Expected Ranking Gain

Maksims Volkovs and Richard Zemel

University of Toronto
IR Learning to Rank: Problem Formulation

• **Input**: a set of $n$ queries $Q = \{q_1, \ldots, q_n\}$

• Each query $q_i$ has a list of documents $D_i = \{d_{i1}, \ldots, d_{im_i}\}$ and a list of relevance levels $L_i = \{l_{i1}, \ldots, l_{im_i}\}$

• The documents are represented as feature vectors in $\mathbb{R}^p$

• Relevance levels typically take small integer values

• **Goal**: returned documents in order of relevance to query

• **Strategy**: create a scoring function $f(q_i, D_i)$ which outputs a set of scores $S_i = \{s_{i1}, \ldots, s_{im_i}\}$ and then sort based on scores to produce ranked list
Standard IR Evaluation Metric: NDCG

- The order “agreement” between $S_i$ and $L_i$ is typically evaluated by NDCG (Normalized Discounted Cumulative Gain).

- Sorting documents according to $S_i$ gives a ranked order:
  $$R_i = \{ r_{ij}, \ldots, r_{im_i} \}$$

  NDCG combines this with relevance levels:
  $$NDCG(R_i, L_i)@T = N_{qi} \sum_{j=1}^{T} \frac{2^{rel(j)} - 1}{\log(1 + j)}$$

  [Breese, Heckerman, Kadie, 1998; Jarvelin & Kekalainen 2000]

- **Problem:** $\frac{\partial NDCG}{\partial f}$ not smooth, makes gradient learning hard.

  Need a good approximation!
Previous Approaches

• **Individual:**
  - directly map features to scores; regression → no relative information between documents
  Prank [Crammer 01]

• **Pairwise:**
  - minimize pairwise misclassification probabilities
  RankNet [Burges 05]; RankBoost [Freund 04]

• **Listwise:**
  - lists of ranked documents as learning instances, minimize list-wise loss function
  LambdaRank [Burges 06]; ListNet [Cao 07]; SoftRank [Taylor 07]; C-CRF [Quin 08]
Two Disadvantages of Current Methods

• **Disadvantage 1**: NDCG is not included directly in the learning objective

• **Disadvantage 2**: higher order document interactions are not explored
  – At inference time the scoring function is always a function of a single document
Our Approach: BoltzRank

• Use a scoring function that depends on individual and pairwise potentials (at training and test time)
• Define a distribution over all possible document rankings
• Use this distribution to get the expectation of the target performance measure
• Maximize the expectation with respect to the scoring function
Scoring Function

• To explore second order interactions make the score for a given document depend on all the other documents:

\[ f(d_j | q, D) = \phi(d_j) + \sum_{k,k\neq j} \varphi(d_j, d_k) \]

• \( \varphi(d_j, d_k) \) allows to enforce learned second order constraints at inference time

• Experimental results show that \( \varphi(d_j, d_k) \) improve ranking accuracy
For a given set of documents $D$ and scores $S$ given by $f$ to $D$ we define a conditional energy of any ranking $R$:

$$E(R|S) = \frac{2}{m \times (m - 1)} \sum_{r_j > r_k} g_q(r_j - r_k)(s_j - s_k)$$

- If $d_j$ is ranked lower than $d_k$ in $R$ then $r_j > r_k$ and $(r_j - r_k) > 0$
- $E(R|S)$ is the lack of compatibility between $R$ and $S$
- We define the conditional probability of:

$$P(R|S) = \frac{1}{Z(S)} \exp(-E(R|S))$$

$$Z(S) = \sum_R \exp(-E(R|S))$$
Learning Objective

- Use $P(R|S)$ to get the expected NDCG:

$$\langle NDCG|S \rangle_P = \sum_{R} P(R|S) NDCG(R, L)$$

- The sum is over exponentially many rank assignments and is intractable

- Use Monte-Carlo estimate instead

$$\langle NDCG|S \rangle_P^{(R_q)} = \sum_{R \in R_q} P^{(R_q)}(R|S) NDCG(R, L)$$

$$P^{(R_q)}(R|S) = \frac{\exp(-E(R|S))}{\sum_{R' \in R_q} \exp(-E(R'|S))}$$
This learning objective allows us to directly incorporate NDCG at any truncation level.

It also allows us to optimize any other IR metric such as Mean Average Precision (MAP) [Baeza-Yates, 1999].

**Problem**: NDCG@T places all emphasis only on the top T documents (example: NDCG@1).

**Solution**: Combine the objective with a less “severe” function.
• We combine the NDCG objective with KL divergence between the true rank distribution $P(R|L)$ and the model's predicted distribution $P(R|S)$:

$$C^{R_q} = - \sum_{R \in R_q} P^{(R_q)}(R|L) \log(P^{(R_q)}(R|S))$$

• The final objective becomes:

$$O^{(R_q)} = \lambda \left< NDCG | S \right>_{P^{R_q}} - (1 - \lambda)C^{(R_q)}$$

• The gradients with respect to $f$ are smooth so can use a straightforward gradient ascent
Learning Objective (cont.)...
Sampling from the model is too expensive so use relevance levels to *pre-compute* sample set for each query:
Experiments: Data

- **OHSUMED**: 106 queries, 16104 query-document pairs
  - 3 relevance levels \{0, 1, 2\}, 45 features per document
- **TD2004**: 75 queries, 75000 query-document pairs
  - Binary relevance levels, 64 features per document
  - Only 1116 relevant documents so subsample irrelevant documents
- Both datasets come with five 60/20/20 splits for training/validation/testing
- Both datasets are part of the newly released LETOR3.0
Experiments: Model Details

• 1-hidden layer neural nets for $\phi$ and $\varphi$
• Input to $\varphi$ is a concatenation of features of the two documents
• Fix the sample size to 100, as found no significant improvement for $> 100$
• Two BoltzRank versions – without $\varphi$ (BoltzRank1) and with $\varphi$ (BoltzRank2)
• Compare to state-of-the-art on both measures: NDCG and MAP
### Results

<table>
<thead>
<tr>
<th>METHOD</th>
<th>NDCG@1</th>
<th>NDCG@2</th>
<th>NDCG@3</th>
<th>NDCG@4</th>
<th>NDCG@5</th>
<th>MAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOLTZRank1</td>
<td>55.43</td>
<td>53.03</td>
<td>51.77</td>
<td><strong>50.26</strong></td>
<td>48.76</td>
<td>45.22</td>
</tr>
<tr>
<td>BOLTZRank2</td>
<td><strong>56.81</strong></td>
<td>54.08</td>
<td><strong>51.83</strong></td>
<td>50.23</td>
<td><strong>49.10</strong></td>
<td><strong>46.04</strong></td>
</tr>
<tr>
<td>AdaRank.NDCG</td>
<td>53.30</td>
<td>49.22</td>
<td>47.90</td>
<td>46.88</td>
<td>46.73</td>
<td>44.98</td>
</tr>
<tr>
<td>AdaRank.MAP</td>
<td>53.88</td>
<td>47.89</td>
<td>46.82</td>
<td>47.21</td>
<td>46.13</td>
<td>44.87</td>
</tr>
<tr>
<td>FRANK</td>
<td>53.00</td>
<td>50.08</td>
<td>48.12</td>
<td>46.94</td>
<td>45.88</td>
<td>44.39</td>
</tr>
<tr>
<td>ListNet</td>
<td>53.26</td>
<td>48.10</td>
<td>47.32</td>
<td>45.61</td>
<td>44.32</td>
<td>44.57</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>METHOD</th>
<th>NDCG@1</th>
<th>NDCG@2</th>
<th>NDCG@3</th>
<th>NDCG@4</th>
<th>NDCG@5</th>
<th>MAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOLTZRank1</td>
<td>45.33</td>
<td>40.00</td>
<td>37.77</td>
<td>36.16</td>
<td>35.91</td>
<td>22.36</td>
</tr>
<tr>
<td>BOLTZRank2</td>
<td>47.67</td>
<td><strong>41.33</strong></td>
<td><strong>39.02</strong></td>
<td><strong>37.57</strong></td>
<td><strong>36.35</strong></td>
<td><strong>23.90</strong></td>
</tr>
<tr>
<td>AdaRank.NDCG</td>
<td>42.67</td>
<td>38.00</td>
<td>36.88</td>
<td>35.24</td>
<td>35.14</td>
<td>19.36</td>
</tr>
<tr>
<td>AdaRank.MAP</td>
<td>41.33</td>
<td>39.33</td>
<td>37.57</td>
<td>36.83</td>
<td>36.02</td>
<td>21.89</td>
</tr>
<tr>
<td>FRANK</td>
<td><strong>49.33</strong></td>
<td>40.67</td>
<td>38.75</td>
<td>35.81</td>
<td>36.29</td>
<td>23.88</td>
</tr>
<tr>
<td>ListNet</td>
<td>36.00</td>
<td>34.67</td>
<td>35.73</td>
<td>34.69</td>
<td>33.25</td>
<td>22.31</td>
</tr>
</tbody>
</table>
Conclusions

- Optimizing proper objective function improves performance
  - Caveat: some form of regularization helps, particularly if the test metric considers only top-ranked item(s)
- Estimating distribution of rankings permits optimization of permutation-based objectives
- Pairwise document information is useful
The End.

Thank You!
What does the pairwise potential learn?

- Plot $\varphi$ weights on the corresponding document features against each other: