# 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 $Q = \{q_1, ...., q_n\}$
- Each query $q_i$ has a list of documents $D_i = \{d_{i1}, ...., d_{im_i}\}$ and a list of relevance levels $L_i = \{l_{i1}, ...., l_{im_i}\}$
- The documents are represented as feature vectors in $\Re^p$
- 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 $S_i = \{s_{i1}, ..., s_{im_i}\}$ 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 (**N**ormalized **D**iscounted **C**umulative **G**ain)
- Sorting documents according to $S_i$ gives a ranked order

$$R_i = \{r_{ij}, ..., r_{im_i}\}$$

NDCG combines this with relevance levels

$$NDCG(R_i, L_i)@T = N_{q_i} \sum_{j=1}^{T} \frac{2^{rel(j)} - 1}{\log(1+j)}$$

[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 $\rightarrow$ 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

# Probability Distribution Over Rankings

- For a given set of documents $D$ and scores $S$ given by $f$ to $D$ we define a conditional energy of any ranking $R$:

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

- 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

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

$$P^{(R_q)}(R|S) = \frac{\exp(-E(R|S))}{\sum_{R' \in R_q} \exp(-E(R'|S))}$$

# Learning Objective (cont.)...

- 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

# Learning Objective (cont.)...

- We combine the NDCG objective with KL divergence between the true rank distribution $P(R|L)$ and the model's predicted distribution $P(R|S)$:

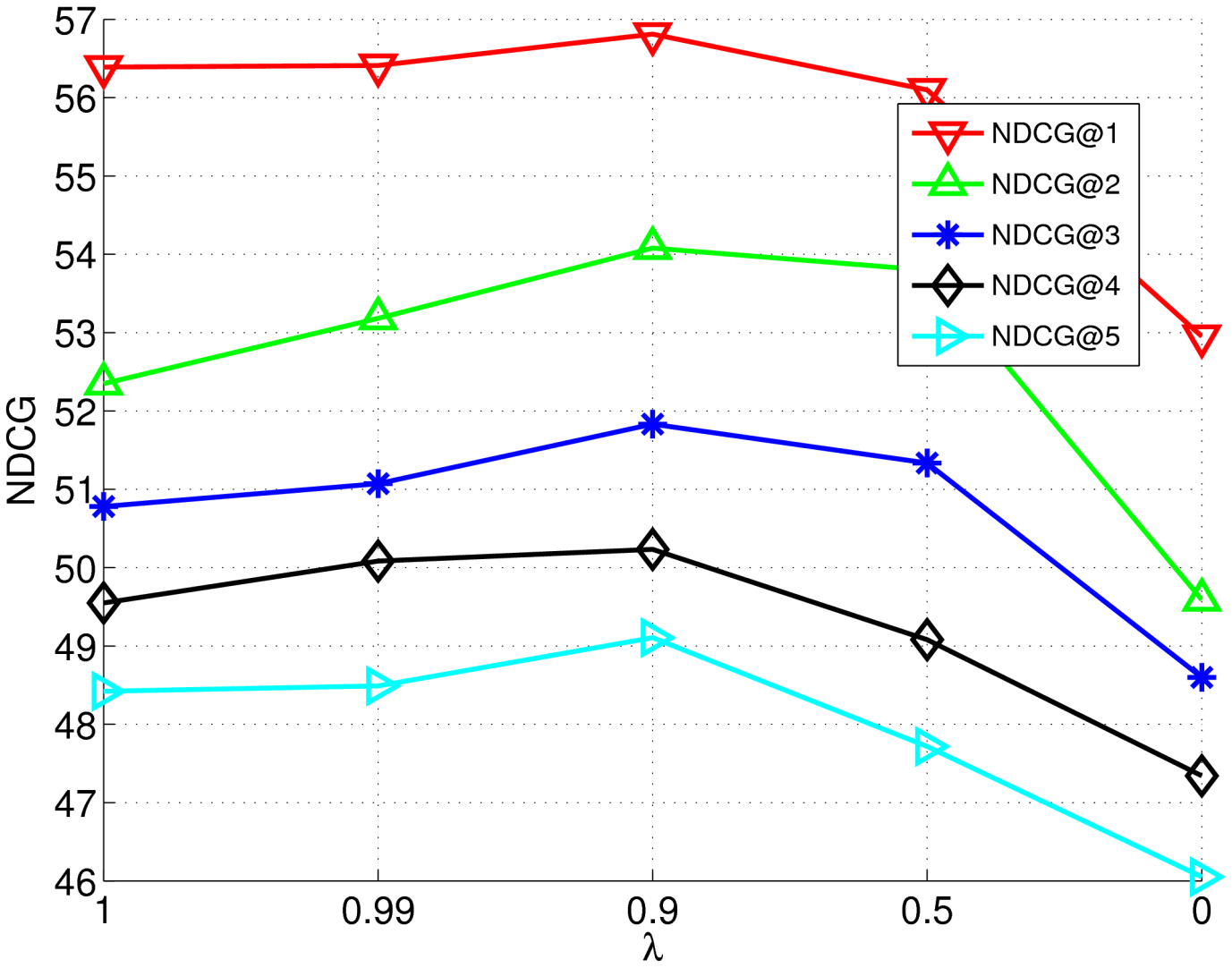
$$C^{R_q} = -\sum_{R \in R_q} P^{(R_q)}(R|L) \log(P^{(R_q)}(R|S))$$

- The final objective becomes:

$$O^{(R_q)} = \lambda \langle NDCG|S \rangle_{P^{R_q}} - (1-\lambda) C^{(R_q)}$$

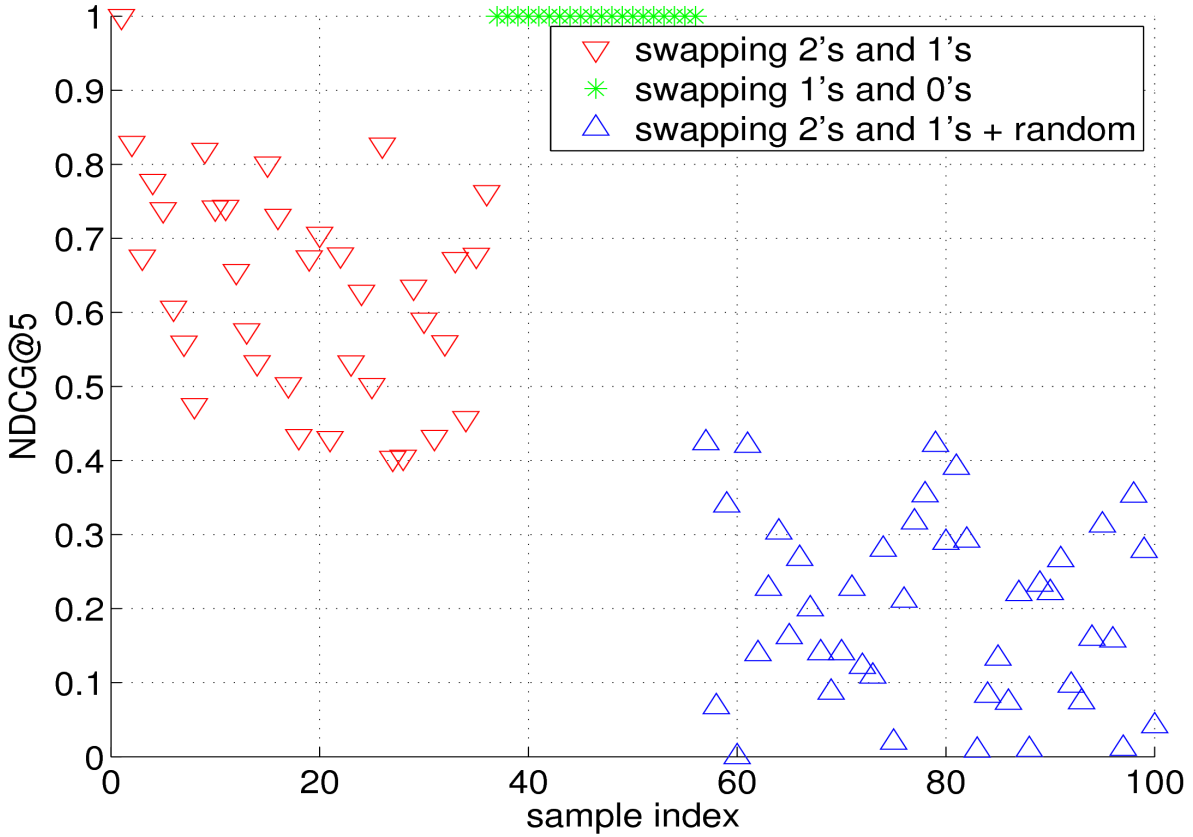
- The gradients with respect to $f$ are smooth so can use a straightforward gradient ascent

# Learning Objective (cont.)...



# Ranking Sample $R_q$

- 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

| Method | NDCG@1 | NDCG@2 | NDCG@3 | NDCG@4 | NDCG@5 | MAP |
|---|---|---|---|---|---|---|
| BoltzRank1 | 55.43 | 53.03 | 51.77 | **50.26** | 48.76 | 45.22 |
| BoltzRank2 | **56.81** | **54.08** | **51.83** | 50.23 | **49.10** | **46.04** |
| AdaRank.NDCG | 53.30 | 49.22 | 47.90 | 46.88 | 46.73 | 44.98 |
| AdaRank.MAP | 53.88 | 47.89 | 46.82 | 47.21 | 46.13 | 44.87 |
| FRank | 53.00 | 50.08 | 48.12 | 46.94 | 45.88 | 44.39 |
| ListNet | 53.26 | 48.10 | 47.32 | 45.61 | 44.32 | 44.57 |
| BoltzRank1 | 45.33 | 40.00 | 37.77 | 36.16 | 35.91 | 22.36 |
| BoltzRank2 | 47.67 | **41.33** | **39.02** | **37.57** | **36.35** | **23.90** |
| AdaRank.NDCG | 42.67 | 38.00 | 36.88 | 35.24 | 35.14 | 19.36 |
| AdaRank.MAP | 41.33 | 39.33 | 37.57 | 36.83 | 36.02 | 21.89 |
| FRank | **49.33** | 40.67 | 38.75 | 35.81 | 36.29 | 23.88 |
| ListNet | 36.00 | 34.67 | 35.73 | 34.69 | 33.25 | 22.31 |

# 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: