Conditional Models for Non-smooth Ranking Loss Functions

Avinava Dubey[1] Jinesh Machchhar[2]
Chiranjib Bhattacharyya[3] Soumen Chakrabarti[4]

[1]IBM Research India
[2]IIT Bombay
[3]IISc Bangalore
[4]IIT Bombay

Ranking

- Given a set of documents and a query, order the documents in such a way that those documents which are relevant to the query is placed above those that are irrelevant
- Not the same as binary classification
- User attention drops off with rank
- Minimize cognitive burden till information need satisfaction
- Has led to many models of ranking loss functions

Overview of learning to rank

- Training data
  - Query set Q
  - Document set D_q associated with each query $q \in Q$
  - Each document $i \in D_q$ is assigned relevance z_{qi}
  - For binary relevance D_q is partitioned into relevant/good documents $D_q^+ \subset D_q$ and irrelevant/bad documents $D_q^- = D_q \setminus D_q^+$
  - Feature vector $x_{qi} \in \mathbb{R}^d$ is constructed from q and doc i
- Training fits a model $w \in \mathbb{R}^d$
- At test time assign score $w^\top x_{ti}$ to each feature vector $x_{ti} \in D_t$ for query t
- Sort by decreasing score, take top 10, say

Structured learning interpretation

- Instead of scoring individual documents ...
- ... think of assigning D_q a permutation or partial order as a classification label
- From a huge label set $\mathcal{Y}$
- In case of permutations, $\mathcal{Y} = (D_q \xrightarrow{1:1} D_q)$
- In case of good/bad partial orders, $y \in \mathcal{Y}_q = \{-1, +1\}^{n_q^+ n_q^-}$, where $n_q^+ = |D_q^+|$ and $n_q^- = |D_q^-|$
- $y_{gb} = +1$ (-1) if document g is ranked above (below) document b
- What's the benefit of thinking this way?

List-oriented loss functions

- y_q is the best $y \in \mathcal{Y}$ for query q
- All good docs ranked above all bad docs
- Loss function $\Delta(y, y_q) \in \mathbb{R}_{\geq 0}$ tells us how bad y is wrt ideal y_q
- Δ can encode what item- or pair-decomposable losses cannot, e.g.:
  - Number of misclassified documents (0/1 additive error)
  - Number of pair inversions (not rank-sensitive)
- Can be exploited to target w to
  - Mean average precision (MAP)
  - Normalized discounted cumulative gain (NDCG)
  - Mean reciprocal rank (MRR)
  - ... and traditional decomposable losses like area under curve (AUC)

Feature map $\phi(x, y)$

- y-cognizant aggregation of document features
- E.g., add vectors from rank 1 through 10 and subtract rest
- More commonly used:

$$\phi_{\text{po}}(x_q, y) = \frac{1}{n_q^+ n_q^-} \sum_{g,b} y_{gb}(x_g - x_b)$$

- At test time, goal is to find $\arg\max_y w^\top \phi(x_q, y)$
- Can be intractable, but sometimes not

Training

- Look for w that minimizes $\sum_q \Delta\left(y_q, \arg\max_y w^\top \phi(x_q, y)\right)$
- Objective not continuous, differentiable, or convex
- Therefore, use convex upper bound hinge loss

$$\min_w \sum_q \max\left\{0, \max_y \Delta_q(y) - w^\top \delta\phi_q(y)\right\}$$

- The bound can be loose
- Only support vectors decide w
- May not always yield a good model

Our contributions

- Parametric conditional probabilistic model $\Pr(y|x; w) \propto \exp(w^\top \phi(x, y))$
- Intuitive minimization of expected ranking loss (unfortunately non-convex) as an alternative to hinge loss
- For specific ϕ and Δ, exact, efficient, closed-form expressions for above optimization
- Convex bound on expected loss objective (unfortunately sums over exponentially many ys)
- Monte-Carlo recipe to sample few ys and still optimize well
- Favorable experimental outcome on LETOR data

Minimizing aggregated expected loss

- Define

$$\Pr(y|x_q; w) = \frac{\exp(w^\top \phi(x_q, y))}{Z_q}$$

  where $Z_q = \sum_{y'} \exp(w^\top \phi(x, y'))$
- Minimize aggregated expected loss

$$\sum_q \sum_y \Pr(y|x_q; w) \Delta(y_q, y)$$

  or log of monotone function of expected loss

$$\sum_q \log \left(\sum_y \Pr(y|x_q; w) f(\Delta(y_q, y)) \right)$$

From loss to gain

- Objective has $\sum_y \cdots$
- Most ys are terrible rankings with $\Delta \to 1$
- Expected loss may drop very slightly on optimization
- To better condition the problem, use gain G rather than loss Δ, e.g., NDCG instead of Δ_{NDCG}

$$\textbf{ExpGain:} \quad \max_w \sum_q \log\left(\sum_y \Pr(y|x_q;w) G_q(y)\right).$$

- Neither loss nor gain optimization is convex
- Beware the sum over y

Polynomial form for AUC and ϕ_{po}

- Notation:

$$S_{qi} = w^\top x_{qi}$$
$$h^q_{gb}(y_{gb}) = \exp\left(\frac{y_{gb}(S_{qg} - S_{qb})}{n^+_q n^-_q}\right)$$

- With gain G being AUC, objective can be written as

$$\sum_q \left\{ \log\left[\sum_y \#1_y \left(\prod_{gb} h^q_{gb}(y_{gb})\right)\right] - \log Z_q \right\}$$

where $\#1_y = \sum_{g,b} [\![y_{gb} = 1]\!]$

Polynomial form for AUC and ϕ_{po} (2)

- Because of $\sum_y \cdots$ we would still take $\sum_q (n_q^+ n_q^-)!$ or $\sum_q 2^{n_q^+ n_q^-}$ time, impractical
- Handy identities (see paper) to replace the $\sum_y \cdots$ with an expression that can be computed in $\sum_q (n_q^+ n_q^-)^2$ time
- Likewise with gradient expression

Toward convexity

- Define $\delta\phi_q(y) = \phi(x_q, y_q) - \phi(x_q, y)$ and consider the distribution

$$\Pr(y|x_q, y_q; w) = \frac{\exp(-w^\top \delta\phi_q(y))}{\sum_{y'} \exp(-w^\top \delta\phi_q(y'))} \tag{1}$$

- Maximum Likelihood Estimate **MLE**

$$L_1(w) = \sum_q L_{1q}(w) = \sum_q \log Z_q(w) \tag{2}$$

Toward convexity (2)

- Expected loss using new distribution

$$L_2(w) = \sum_q L_{2q}(w) = \sum_q \mathbb{E}_{Y \sim (1)}(\Delta_q(Y)) \quad (3)$$

- Finally consider another distribution

$$\Pr(y|x_q, y_q) = \frac{\exp(-w^\top \delta\phi_q(y) + \Delta_q(y))}{\sum_{y'} \exp(-w^\top \delta\phi_q(y') + \Delta_q(y'))} \quad (4)$$

Toward convexity (3)

- ... and corresponding **ConvexLoss:**

$$L(w) = \sum_q \log \sum_{y'} \exp(-w^\top \delta\phi_q(y') + \Delta_q(y')) \tag{5}$$

- Can find w to minimize ConvexLoss
- Problem: $\sum_y \cdots$ is back
- Anyway we want to go beyond AUC and ϕ_{po}

Markov Chain Monte Carlo sampling

while not enough samples **do**
  (re)start a random walk at a well-chosen state y^0
  for some number of steps $t = 1, 2, \ldots$ **do**
    transition from y^{t-1} to y^t
    make an accept/reject decision on y^t
  collect some subset of $y^0, y^1, y^2, \ldots$ as samples

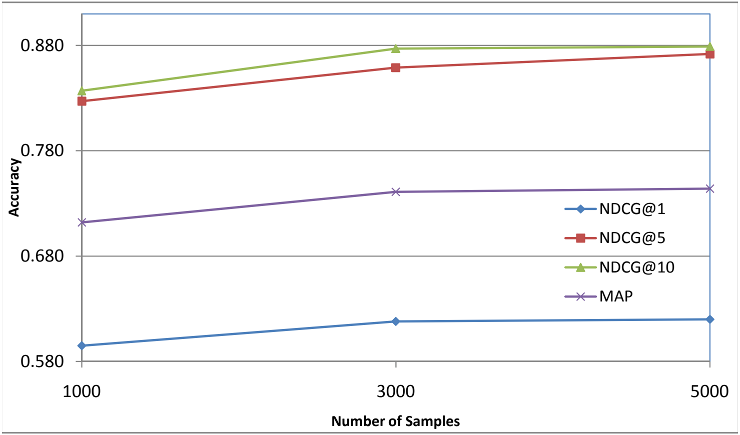
- Choose restart states to span a variety of Δs
- In each walk, make local changes in y so as to stay near to the restart Δ
- New swap method to make transitions (see paper)

Experiments: Accuracy

- All data sets from LETOR
- Baselines RankSVM, SVMmap, RankBoost
- Example accuracy comparison on HP2004:

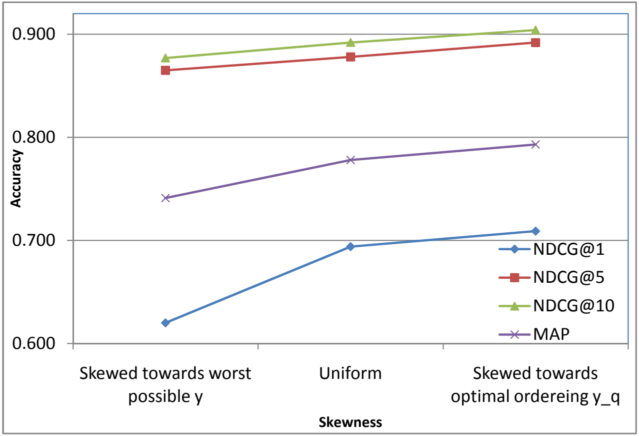
	NDCG@1	NDCG@5	NDCG@10	MAP
SVMmap	0.665	0.835	0.845	0.746
Rankboost	0.653	0.821	0.845	0.739
RankSVM	0.695	0.852	0.877	0.764
MLE	0.665	0.854	0.872	0.755
ExpGain_AUC	0.695	0.862	0.885	0.774
ExpGain_MAP	0.680	0.875	0.895	0.775
ExpGain_NDCG	0.680	0.872	0.890	0.772
L_3 AUC	0.695	0.880	0.898	0.771
L_3 MAP	0.709	0.883	0.900	0.786
L_3 NDCG	0.681	0.885	0.900	0.773
ConvexLoss_AUC	0.682	0.890	0.905	0.778
ConvexLoss_MAP	0.724	0.874	0.885	0.791
ConvexLoss_NDCG	0.709	0.892	0.904	0.793

Accuracy vs. MCMC samples



- More samples $\Longrightarrow$ better accuracy
- Saturates long before $\mathcal{Y}$ is approached

Effect of restart skew



- Two restart points, $\Delta = 0, \Delta_{\mathsf{max}} \approx 1$
- Skewing toward best y is better

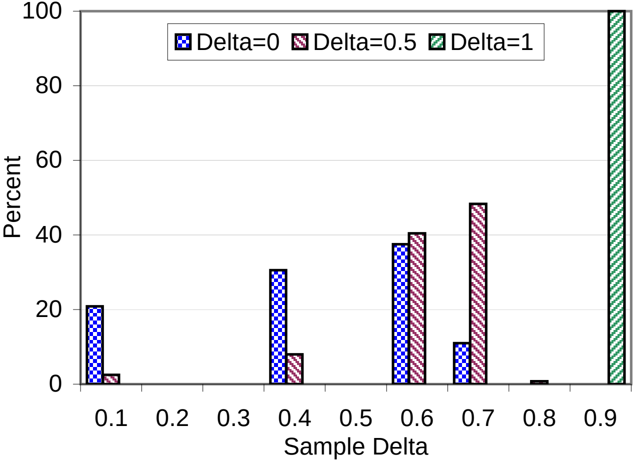
Effect of restart skew (2)

- Potentially surprising, given learning needs both good and bad examples to learn
- Sample y w.p. $\propto \exp(k\Delta)$
- $k \ll 0$ means favor $\Delta \to 0$, better

$k \to$	5	0	-5	-10	-20
MAP	.033	.089	.706	.774	.827
NDCG@10	.041	.124	.815	.905	.918

- Why?

Effect of restart skew (3)



- Three restart seeds

Effect of restart skew (4)

- Perturbing y with $\Delta = 0$ can push it far, say, $\Delta \approx 0.7$ quite easily
- If y has $\Delta \approx 1$, most perturbations will keep Δ near 1
- There algo will not see enough good ys
- Which is why skewing restarts toward $\Delta \approx 0$ is good

Conclusion

- Conditional probabilistic model $\Pr(y|x; w) \propto \exp(w^\top \phi(x, y))$ for ranking
- Challenges: $\sum_{y \in \mathcal{Y}} \cdots$, nonconvexity
- Efficient closed form for specific but widely-used ϕ and Δ
- Convex upper bound to expected loss in all cases
- Monte-Carlo sampling method to avoid $\sum_{y \in \mathcal{Y}} \cdots$
- Competitive accuracy in experiments
- In particular, generally better than
  - Hinge loss with max-margin learning techniques
  - Boosting with weak learners
- Training by sampling the space of ys this way may have broader application