Structured Learning for Non-Smooth Ranking Losses

Soumen Chakrabarti

With Rajiv Khanna, Uma Sawant, Chiru Bhattacharyya

Learning to rank: Training, testing

- A set of queries
- Each query q comes with a set of documents
- Each doc represented as a feature vector $x_{qi} \in \mathbb{R}^d$; $d \approx 50 \ldots 300$
- Doc x_{qi} may be good (relevant) or bad (irrelevant) wrt q: $z_{qi} \in \{0, 1\}$
- n_q^+ good docs D_q^+; n_q^- bad docs D_q^-
- Learner estimates model $w \in \mathbb{R}^d$
- During testing, good/bad not known
- Score of doc is dot product $f_w(x_{qi}) = w^\top x_{qi}$
- Sort docs by decreasing score, present top-k

Loss functions

- Good doc index g, bad doc index b
- Ideal w ensures $f(x_{qg}) > f(x_{qb})$ for all g, b
- If not possible, which of many imperfect ws should we pick?
- Depends on design of loss function

  Elementwise: Charge for regression error:
  $$\sum_i \big(f(x_{qi}) - z_{qi}\big)^2$$
  Pairwise: Charge for wrong pair orderings:
  $$\sum_{g,b} [\![f(x_{qg}) < f(x_{qb})]\!]$$
  Listwise: Loss is a function of ideal ordering and sorted order defined by scores $f(x_{qi})$

Listwise loss function

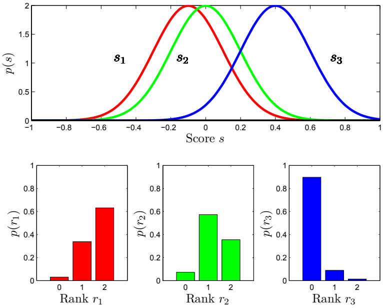
- $x_q \in \mathcal{X}_q$: all document vectors for query q
- $\mathcal{Y}_q$: space of total or partial orders
  - y is a permutation: $|\mathcal{Y}_q| = (n_q^+ + n_q^-)!$
  - $y_{gb} = \begin{cases} -1, & \text{if } g \text{ after } b \\ +1, & \text{if } g \text{ before } b \end{cases}$ — $|\mathcal{Y}_q| = 2^{n_q^+ n_q^-}$
- y_q^*: perfect ranking for query q (all good before any bad; order among good or bad unimportant)
- y: some other total or partial order on x_qs
- General loss function $\Delta(y_q^*, y)$

⊕ Can express reward for good docs at top ranks

⊖ Rank known only via sort, ∴ loss not continuous, differentiable or convex in w

Non-smooth loss: Earlier efforts

- Bound by elementwise regression loss (McRank)
- Bound by pairwise hinge loss $\sum_{i \succ j} \max\{0, 1 - f(x_i) + f(x_j)\}$ (RankSVM)
- Pairwise loss weighted by function of current ranks (LambdaRank)
- Probability distribution over rankings (ListNet)
- Model $f(x_i)$ as mean of normal score distribution, map scores to expected ranks (SoftRank)



Listwise feature map $\phi(x_q, y) \in \mathbb{R}^d$

- Rank-sensitive aggregation of doc feature vectors

$$\text{E.g.,} \qquad \phi_{\text{po}}(x, y) = \sum_{g,b} y_{gb}(x_g - x_b)$$

(intuition: want $y_{gb} = +1$ and $w^\top x_g > w^\top x_b$)
- When testing, predict $\arg\max_y w^\top \phi(x_q, y)$
- For ϕ_{po}, equivalent to sort by decreasing $w^\top x_{qi}$
- For training, find w so that, $\forall q$, $\forall y \neq y_q^*$:

$$w^\top \phi(x_q, y_q^*) + \xi_q \geq \Delta(y_q^*, y) + w^\top \phi(x_q, y)$$

- Usual SVM objective $w^\top w + C \sum_q \xi_q$

Cutting plane algorithm overview

- Problem: Exponential number of constraints
- Begin with *no* constraints and find w
- Look for violators

$$w^\top \phi(x_q, y_q^*) + \xi_q + \epsilon < \underbrace{\Delta(y_q^*, y) + w^\top \phi(x_q, y)}_{\text{maximize this}}$$

- Add these to the set of constraints and repeat
- For fixed ϵ, Tsochanteridis+ showed that a constant number of rounds give ϵ-approximate solution

Loss-augmented argmax: NDCG

- Recall $z_{qi} = 0$ for bad, 1 for good doc
- Rank discount $D(r)$ decreases with rank r
- $y[i]$ = doc at rank i under permutation y
- y^* puts all good docs at top ranks

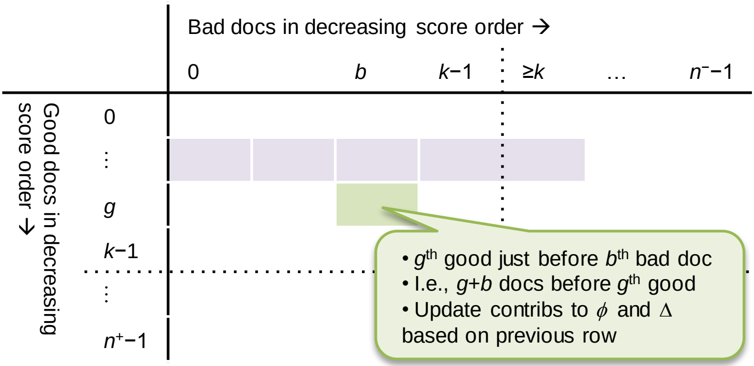
$$\begin{aligned} \mathrm{DCG}(y) &= \sum_{0 \le i < k} z_{q,y[i]} D(i) \\ \mathrm{NDCG}(y) &= \mathrm{DCG}(y) / DCG(y^*) \\ \Delta_{\mathrm{ndcg}}(y^*, y) &= 1 - \mathrm{NDCG}(y) \end{aligned}$$

Contribution: Simple, $O(n_q \log n_q)$-time argmax routine for ϕ_{po} and Δ_{ndcg}, leading to SVMNDCG

Generic template to max $w^\top \phi + \Delta$

- Assume two levels of relevance $z_{qi} \in \{0, 1\}$
- Δ unchanged if two good (or bad) docs swapped

$\therefore$ There exists an optimal y that can be formed by merging good and bad in decreasing score order



Is training on "true" Δ always best?

	OHSUMED			TD2003			TD2004			TREC2000			TREC2001		
	MRR10	NDCG10	MAP	MRR10	NDCG10	MAP	MRR10	NDCG10	MAP	MRR10	NDCG10	MAP	MRR10	NDCG10	MAP
MRR	0.80	0.62	0.57	0.63	0.41	0.33	0.63	0.44	0.38	0.67	0.41	0.24	0.64	0.43	0.23
NDCG*	0.82	0.64	0.58	0.60	0.40	0.31	0.61	0.49	0.40	0.69	0.46	0.27	0.62	0.44	0.26
DORM	0.81	0.64	0.58	0.59	0.36	0.29	0.47	0.34	0.30	0.66	0.41	0.24	0.62	0.44	0.25
MAP	0.81	0.64	0.59	0.62	0.41	0.31	0.61	0.50	0.41	0.70	0.47	0.28	0.64	0.45	0.27

MRR: Max mean reciprocal rank of #1 good doc

NDCG: Maximize NDCG

DORM: Ditto; Hungarian docs-to-ranks assignment (Chapelle+ 2007)

MAP: Maximize mean average precision (Yue+ 2007)

- Observation: Best test accuracy for a given criterion may be obtained with a *different* Δ during training!
- Mismatch between ϕ and Δ make constraints hard to satisfy except with large slacks ξ_q

What use is a perfect loss function, if no matching feature map is to be found?

Tailoring ϕ to Δ: MRR

- $\phi_{\text{po}}(x, y) = \sum_{g,b} y_{gb}(x_g - x_b)$ looks symmetric across good-bad pairs
- ϕ_{po} can also be written as $\sum_g \sum_{b:b \succ g}(x_g - x_b)$
- Let r_1 be rank of first good doc
- (Roughly speaking) $\Delta_{\text{mrr}} = 1 - 1/r_1$
- I.e., no credit for 2nd and subsequent good docs
- $\phi(x, y)$ should only focus on first good doc
- Accordingly, we define

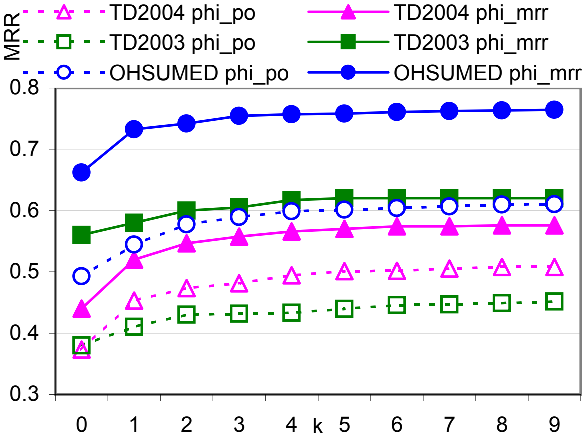
$$\phi_{\text{mrr}}(x, y) = \sum_{b:b \succ g_0(y)} (x_b - x_{g_0(y)}),$$

where $g_0(y)$ is the first good doc in ordering y

Modified $\arg\max_y w^\top \phi_{\text{mrr}} + \Delta_{\text{mrr}}$ algo

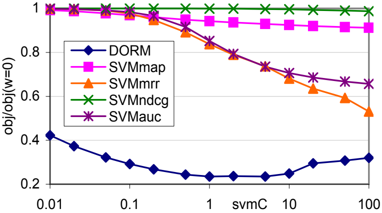
- $1, 1/2, 1/3, 1/k, 0$ only possible values of Δ_{mrr}
- For a given value of MRR, say $1/r$, first good doc must be at rank r
- For a given configuration $\underbrace{b, \ldots, b}_{r-1}, \underbrace{g}_{r}, \underbrace{?, ?, \ldots}_{\text{rest}}$ need to fill good and bad slots to maximize $w^\top \phi$
- Bad docs b at $1, \ldots, r-1$ with largest $w^\top x_b$
- Good doc g with smallest $w^\top x_g$ at position r
- Add up Δ and $w^\top \phi$ for each possible Δ and take maximum
- (MRR $= 0$ handled separately)

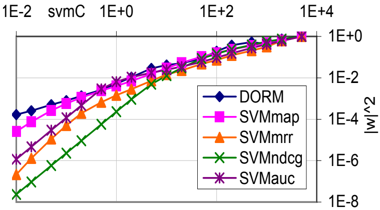
Benefits of using ϕ_{mrr} with Δ_{mrr}



- ϕ_{mrr} far superior to ϕ_{po} (originally used for AUC)
- No ϕ_{ndcg} found yet ☹

Optimization health





- $w = \vec{0}$ is always a (useless) solution
- We broke down a nasty optimization into a convex QP and a simple argmax problem
- How much can we reduce the objective compared to $w = \vec{0}$ as we increase C?
- How does $\|w\|_2$ grow with C?

What use is a library of perfect loss functions, if we have no idea which Δ users want?

- MRR suited for navigational queries
- NDCG suited for researching a topic
- Both kinds of queries very common
- Must hedge our bets

Train for multiple Δs: SVMCOMBO

- Can a single w to do well for many Δs?

$$\arg\min_{w;\xi\geq\vec{0}} w^\top w + \sum_\ell C_\ell \frac{1}{|Q|}\sum_q \xi_q^\ell \quad \text{s.t.}$$

$$\forall \ell, q, \forall y \neq y_q^* : w^\top \delta\phi_q(y) \geq \Delta_\ell(y_q^*, y) - \xi_q^\ell$$

ℓ ranges over loss types NDCG, MRR, MAP, ...

- Empirical risk (training error) $R(w,\Delta) = \frac{1}{|Q|}\sum_q \Delta(y_q^*, f_w(x_q))$
- Can show

$$\boxed{\sum_\ell C_\ell \frac{1}{|Q|}\sum_q \xi_q^\ell} \geq \sum_\ell R(w,\Delta_\ell) \geq R(w, \max_\ell \Delta_\ell)$$

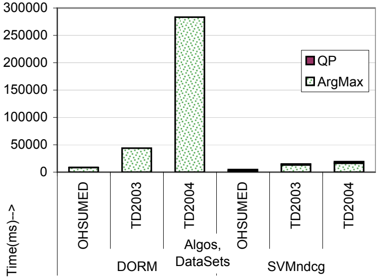
- I.e. learning minimizes upper bound on worst loss

Test accuracy vs. training loss function

	OHSUMED			TD2003			TD2004			TREC2000			TREC2001		
	MRR10	NDCG10	MAP	MRR10	NDCG10	MAP	MRR10	NDCG10	MAP	MRR10	NDCG10	MAP	MRR10	NDCG10	MAP
AUC	.799	.635	.582	.510	.349	.256	.639	.501	.420	.607	.448	.267	.632	.441	.264
MAP	.808	**.642**	**.586**	.618	.411	*.314*	.614	.496	*.412*	**.696**	**.469**	**.277**	.636	**.450**	**.272**
NDCG	.790	*.636*	.581	.587	*.372*	.302	.631	*.457*	.374	.517	*.323*	.175	.608	*.356*	.171
NDCG-NC	**.818**	.640	.582	.595	.404	.306	.611	.486	.404	.685	.455	.265	.624	.443	.264
MRR	*.795*	.623	.570	*.628*	.405	.330	*.629*	.441	.383	*.670*	.410	.244	*.643*	.426	.230
COMBO	.813	.635	.578	**.667**	**.434**	**.345**	**.647**	.458	.384	.695	.465	**.277**	**.647**	.449	**.272**
DORM	.807	.637	.583	.587	.362	.290	.474	.340	.297	.662	.413	.243	.621	.435	.250
McRank	.701	.565	.527	.650	.403	.232	.588	**.529**	**.453**						

- Row: training Δs, column: test criterion
- SVMCOMBO, SVMMAP good across the board
- Did not tune C_ℓ yet
- Listwise Δs better than elementwise or pairwise

SVMNDCG speed and scalability



Dataset	McRank tree	McRank boost	McRank total	SVMNDCG	SVMMRR
OHSUMED	1034	67	1102	4.8	30.6
TD2003	9730	383	10113	14.9	125
TD2004	8760	548	9308	19.1	148

SVMCOMBO is

- 15× faster than DORM
- 100× faster than McRank

while being more accurate in over 75% of data sets

Takeaway

- New efficient learners for MRR and NDCG
- Asserting the “correct” Δ may not be best
- Satisfy multiple Δs using SVMCOMBO
- Listwise structured ranking is faster
- And frequently more accurate than competition

Future work

- Design ϕs better tailored to respective Δs
- Evaluate on larger data sets
- Diversity and bypass rates
- Is convexity overrated?