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 \( x_{qi} \in \mathbb{R}^d; d \approx 50\ldots300 \)
- 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 $w$s should we pick?
- Depends on design of loss function

**Elementwise**: Charge for regression error:

\[ \sum_i (f(x_{qi}) - z_{qi})^2 \]

**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

- $x_q \in X_q$: 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 $x_q$s
- General loss function $\Delta(y^*_q, y)$
  - Can express reward for good docs at top ranks
  - Rank known only via sort, $\therefore$ loss not continuous, differentiable or convex in $w$
Non-smooth loss: Earlier efforts

- Bound by elementwise regression loss (**McRank**)
- Bound by pairwise hinge loss
  \[ \sum_{i > j} \max\{0, 1 - f(x_i) + f(x_j)\} \]
  (**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

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

(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 $\phi_{po}$, 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 $\epsilon$, Tsochanteridis+ showed that a constant number of rounds give $\epsilon$-approximate solution
Loss-augmented argmax: NDCG

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

\[
\text{DCG}(y) = \sum_{0 \leq i < k} z_{q,y[i]} D(i)
\]
\[
\text{NDCG}(y) = \frac{\text{DCG}(y)}{\text{DCG}(y^\ast)}
\]
\[
\Delta_{\text{ndcg}}(y^\ast, y) = 1 - \text{NDCG}(y)
\]

Contribution: Simple, $O(n_q \log n_q)$-time argmax routine for $\phi_{po}$ and $\Delta_{\text{ndcg}}$, leading to SVM$_{\text{NDCG}}$
Generic template to max $w^T \phi + \Delta$

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

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

<table>
<thead>
<tr>
<th>Good docs in decreasing score order</th>
<th>Bad docs in decreasing score order</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n^+1$</td>
<td>$n^-1$</td>
</tr>
<tr>
<td>$k-1$</td>
<td>$k-1$</td>
</tr>
<tr>
<td>$g$</td>
<td>$g$</td>
</tr>
<tr>
<td>$b$</td>
<td>$b$</td>
</tr>
<tr>
<td>$0$</td>
<td>$0$</td>
</tr>
</tbody>
</table>

- $g^{th}$ good just before $b^{th}$ bad doc
- I.e., $g+b$ docs before $g^{th}$ good
- Update contribs to $\phi$ and $\Delta$ based on previous row
Is training on “true” $\Delta$ always best?

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MRR10</td>
<td>NDCG10</td>
<td>MAP</td>
<td>MRR10</td>
<td>NDCG10</td>
</tr>
<tr>
<td>MRR</td>
<td>0.80</td>
<td>0.62</td>
<td>0.57</td>
<td>0.63</td>
<td>0.41</td>
</tr>
<tr>
<td>NDCG*</td>
<td>0.82</td>
<td>0.64</td>
<td>0.58</td>
<td>0.60</td>
<td>0.40</td>
</tr>
<tr>
<td>DORM</td>
<td>0.81</td>
<td>0.64</td>
<td>0.58</td>
<td>0.59</td>
<td>0.36</td>
</tr>
<tr>
<td>MAP</td>
<td>0.81</td>
<td>0.64</td>
<td>0.59</td>
<td>0.62</td>
<td>0.41</td>
</tr>
</tbody>
</table>

**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 $\Delta$ during training!

Mismatch between $\phi$ and $\Delta$ make constraints hard to satisfy except with large slacks $\xi_q$

What use is a perfect loss function, if no matching feature map is to be found?
Tailoring $\phi$ to $\Delta$: MRR

- $\phi_{po}(x, y) = \sum_{g,b} y_{gb}(x_g - x_b)$ looks symmetric across good-bad pairs
- $\phi_{po}$ 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) $\Delta_{mrr} = 1 - 1/r_1$
- I.e., no credit for 2nd and subsequent good docs
- $\phi(x, y)$ should only focus on first good doc
- Accordingly, we define

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

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 \( \Delta_{\text{mrr}} \)
- For a given value of MRR, say 1/r, first good doc must be at rank \( r \)
- For a given configuration \( b, \ldots, b, \quad g, \quad ?, \quad ?, \quad \ldots \) \( r-1 \quad r \quad \text{rest} \)
  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 \( \Delta \) and \( w^\top \phi \) for each possible \( \Delta \) and take maximum
- (MRR = 0 handled separately)
Benefits of using $\phi_{mrr}$ with $\Delta_{mrr}$

- $\phi_{mrr}$ far superior to $\phi_{po}$ (originally used for AUC)
- No $\phi_{ndcg}$ 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 $\Delta$ 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 $\Delta$s: SVMcombo

- Can a single $w$ to do well for many $\Delta$s?

$$\arg\min_{w;\xi \geq 0} w^T w + \sum\ell C_{\ell} \frac{1}{|Q|} \sum_q \xi_q$$

s.t.

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

$l$ 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

$$\sum\ell C_{\ell} \frac{1}{|Q|} \sum_q \xi_q \geq \sum\ell R(w, \Delta_{\ell}) \geq R(w, \max\ell \Delta_{\ell})$$

- I.e. learning minimizes upper bound on worst loss
### Test accuracy vs. training loss function

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MRR10</td>
<td>NDCG10</td>
<td>MAP</td>
<td>MRR10</td>
<td>NDCG10</td>
</tr>
<tr>
<td>AUC</td>
<td>.799</td>
<td>.635</td>
<td>.582</td>
<td>.510</td>
<td>.349</td>
</tr>
<tr>
<td>MAP</td>
<td>.808</td>
<td><strong>.642</strong></td>
<td>.586</td>
<td>.618</td>
<td>.411</td>
</tr>
<tr>
<td>NDCG</td>
<td>.790</td>
<td>.636</td>
<td>.581</td>
<td>.587</td>
<td>.372</td>
</tr>
<tr>
<td>NDCG-NC</td>
<td><strong>.818</strong></td>
<td>.640</td>
<td>.582</td>
<td>.595</td>
<td>.404</td>
</tr>
<tr>
<td>MRR</td>
<td>.795</td>
<td>.623</td>
<td>.570</td>
<td>.628</td>
<td>.405</td>
</tr>
<tr>
<td>COMBO</td>
<td>.813</td>
<td>.635</td>
<td>.578</td>
<td><strong>.667</strong></td>
<td><strong>.434</strong></td>
</tr>
<tr>
<td>DORM</td>
<td>.807</td>
<td>.637</td>
<td>.583</td>
<td>.587</td>
<td>.362</td>
</tr>
<tr>
<td>McRank</td>
<td>.701</td>
<td>.565</td>
<td>.527</td>
<td>.650</td>
<td>.403</td>
</tr>
</tbody>
</table>

- **Row**: training $\Delta$s, column: test criterion
- **SVMcombo**, **SVMmap** good across the board
- Did not tune $C_\ell$ yet
- Listwise $\Delta$s better than elementwise or pairwise
**SVMnDCG** speed and scalability

![Graph showing time comparison between SVMnDCG and other algorithms]

- **SVMcombo** is
  - 15x faster than **DORM**
  - 100x faster than **McRank**

while being more accurate in over 75% of data sets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>McRank tree</th>
<th>McRank boost</th>
<th>McRank total</th>
<th>SVMnDCG</th>
<th>SVMmrr</th>
</tr>
</thead>
<tbody>
<tr>
<td>OHSUMED</td>
<td>1034</td>
<td>67</td>
<td>1102</td>
<td>4.8</td>
<td>30.6</td>
</tr>
<tr>
<td>TD2003</td>
<td>9730</td>
<td>383</td>
<td>10113</td>
<td>14.9</td>
<td>125</td>
</tr>
<tr>
<td>TD2004</td>
<td>8760</td>
<td>548</td>
<td>9308</td>
<td>19.1</td>
<td>148</td>
</tr>
</tbody>
</table>
Takeaway

- New efficient learners for MRR and NDCG
- Asserting the “correct” $\Delta$ may not be best
- Satisfy multiple $\Delta$s using SVMcombo
- Listwise structured ranking is faster
- And frequently more accurate than competition

Future work

- Design $\phi$s better tailored to respective $\Delta$s
- Evaluate on larger data sets
- Diversity and bypass rates
- Is convexity overrated?