LEARNING TO RANK: FROM PAIRWISE APPROACH TO LISTWISE APPROACH

Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, Hang Li ,
ICML'2007

PPT Edited：

Motivation:
Ranking is the central issues of many applications.

- Learning to rank is useful for document retrieval, collaborative filtering, and many other applications.
- Take document retrieval as example： Assume that there is a collection of documents. The task is follows.
  - *given a query, the **ranking function** assigns a score to each document*
  - *ranks the documents in descending order of the scores (represents the relevance of documents)*
  - *Learning: Get the **ranking function** to precisely predict the ranking lists*
- learning to rank has been drawing attention in the ***machine learning***

Related Work : Learning based on pairwise approach

- Basic idea:
  for documents pair $< d_x, d_y >$, ***classified*** *as* $d_x > d_y$ *or* $d_y > d_x$

- Examples:
  - *Ranking SVM: Herbrich et al.(1999),*
  - *RankBoost: Freund et al.(1998),*
  - *Neural Network: Burges et al.(2005).*

- Other approaches

- ***probability models*** for representing ranking lists of objects
  - Luce (1959), Plackett (1975)

Related Work:
Learning based on pairwise approach

- Advantages:
  - *existing methodologies on classification can be directly applied*
  - *the training data can be easily obtained*
- Disadvantages:
  - *minimizing errors in classification of document pairs, rather than in ranking*
  - *Assumption that the document pairs are generated i.i.d. is too strong*
  - *the number of generated*
  - *numbers of document pairs varies largely and models will bias toward queries with more document pairs(Table 2)*

Table 2. Document-pair number distribution

Pair Number	Query Number
<5000	61
<10000	29
<15000	8
<20000	6
>=20000	2

Listwise Approach: Basic Idea

- use ***documents list*** as instances in learning
- listwise loss function based on two *probability models*
  - *permutation probability*
  - *top k probability*
- model :Neural Network
- algorithm: Gradient Descent

Listwise Approach: Notation

- queries set: $Q = \{q^{(1)}, q^{(2)}, \dots, q^{(m)}\}$
- documents set associated with $q^{(i)}$: $d^{(i)} = \left(d_1^{(i)}, d_2^{(i)}, \dots, d_{n^{(i)}}^{(i)}\right)$
  - *size of set :* $n^{(i)}$
- target scores associated with $d^{(i)}$: $y^{(i)} = \left(y_1^{(i)}, y_2^{(i)}, \dots, y_{n^{(i)}}^{(i)}\right)$
- feature vector: $x_j^{(i)} = \Psi\left(q^{(i)}, d_j^{(i)}\right)$

which created from query-document pair $\left(q^{(i)}, d_j^{(i)}\right), i = 1,2 \dots, m; j = 1,2 \dots, n^{(I)}$

- list of features: $x^{(i)} = \left(x_1^{(i)}, \dots, x_{n^{(i)}}^{(i)}\right)$
- list of target scores: $y^i = \left(y_1^i, \dots, y_{n^{(i)}}^{(i)}\right)$
- training set: $\tau = \left\{x^{(i)}, y^{(i)}\right\}_{i=1}^{m}$
- ranking function: f
  - *output for* $x_i^{(j)}$ *(corresponding to* $d_j^{(i)}$*)* : $f\left(x_j^{(i)}\right)$
- predicted scores list: $z^{(i)} = \left(f\left(x_1^{(i)}\right), \dots, f\left(x_{n^{(i)}}^{(i)}\right)\right)$
- total losses: $\sum_{i=1}^{m} L\left(y^{(i)}, z^{(i)}\right)$ where L is a listwise loss function.

Listwise Approach: Probability Models

- Permutation Probability
  - *the probability of permutation π given the list of scores s*

$$P_s(\pi) = \prod_{j=1}^{n} \frac{\phi(s_{\pi(j)})}{\sum_{k=j}^{n} \phi(s_{\pi(k)})}$$

  - *notation:*
    - π is a permutation of n numbers corresponding to n documents
    - $\pi(j)$ is the j^{th} object(document) in permutation π
    - s is a scores list, where $s_{\pi(j)}$ is the score of j^{th}objest in π
      - *s is given by y or z*
    - $\phi(.)$ is an increasing and strictly positive function

Listwise Approach: Probability Models

- Permutation Probability(con'd)
  - *the probability of permutation π given the list of scores s:*

$$P_s(\pi) = \prod_{j=1}^{n} \frac{\phi(s_{\pi(j)})}{\sum_{k=j}^{n} \phi(s_{\pi(k)})}$$

  - *In fact, only when the objects with larger scores ranked ahead of other objects in the permutation will the probability of the resulting permutation be higher.*

Listwise Approach: Top k Probability

- disadvantage of Permutation Probability:

  calculation is too complex $O(n!)$for n documents

- Top k Probability:
  - *only take top k positions into consideration, ignoring the rest.*
  - *there are **in total** $\frac{n!}{(n-k)!}$**elements** in top k subgroup $g_k(j_1, j_2, \dots, j_k)$ in which all the elements are permutation π having the same top k objetcs.*

$$P_s(\mathcal{G}_k(j_1, j_2, \dots, j_k)) = \sum_{\pi \in \mathcal{G}_k(j_1, j_2, \dots, j_k)} P_s(\pi) = \prod_{t=1}^{k} \frac{\phi(s_{j_t})}{\sum_{l=t}^{n} \phi(s_{j_l})}$$

  - *the objects with **larger scores** ranked **ahead** in the k objects to promote a higher probability*

Listwise Approach: Listwise loss function

- Define the metric between the top *k* ***probability distributions*** corresponding to the two given lists of scores as the listwise loss function (between two scores list).
- use ***Cross Entropy*** as metric:

$$L(y^{(i)}, z^{(i)}) = -\sum_{\forall g \in \mathcal{G}_k} P_{y^{(i)}}(g) \log(P_{z^{(i)}}(g))$$

Listwise Approach: ListNet

- Learning method:

  Neural Network as model and Gradient Descent as optimization algorithm

- With Cross Entropy as metric, the loss for query $q^{(i)}$ becomes

$$L(y^{(i)}, z^{(i)}(f_\omega)) = -\sum_{\forall g \in \mathcal{G}_k} P_{y^{(i)}}(g) \log(P_{z^{(i)}(f_\omega)}(g))$$

- The gradient of $L\left(y^{(i)}, z^{(i)}(f_\omega)\right)$ with respect to parameter ω can be calculated as follows

$$\Delta\omega = \frac{\partial L(y^{(i)}, z^{(i)}(f_\omega))}{\partial\omega} = -\sum_{\forall g \in \mathcal{G}_k} \frac{\partial P_{z^{(i)}(f_\omega)}(g)}{\partial\omega} \frac{P_{y^{(i)}}(g)}{P_{z^{(i)}(f_\omega)}(g)}$$

Listwise Approach: ListNet(con'd)

Algorithm 1 Learning Algorithm of ListNet

Input: training data $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\}$
Parameter: number of iterations T and learning rate η
Initialize parameter ω
for $t = 1$ **to** T **do**
  for $i = 1$ **to** m **do**
    Input $x^{(i)}$ of query $q^{(i)}$ to Neural Network and compute score list $z^{(i)}(f_\omega)$ with current ω
    Compute gradient $\Delta\omega$ using Eq. (5)
    Update $\omega = \omega - \eta \times \Delta\omega$
  end for
end for
Output Neural Network model ω

Experimental Results

- dataset : TREC, OHSUMED, CSearch
- evaluation:Normalized Discounted Cumulative Gain (NDCG), Mean Average Precision (MAP)
- methods:ListNet, RankNet, Ranking SVM, RankBoost

Experimental Results: Accuracy

- ListNet ***outperforms*** the three baseline methods(on pair approaches) of RankNet, Ranking SVM, and RankBoost in terms of all measures on all datasets.

Table 1. Ranking accuracies in terms of MAP

ALGORITHMS	LISTNET	RANKBOOST	RANKSVM	RANKNET
TREC	0.216	0.174	0.193	0.197
OHSUMED	0.305	0.297	0.297	0.303

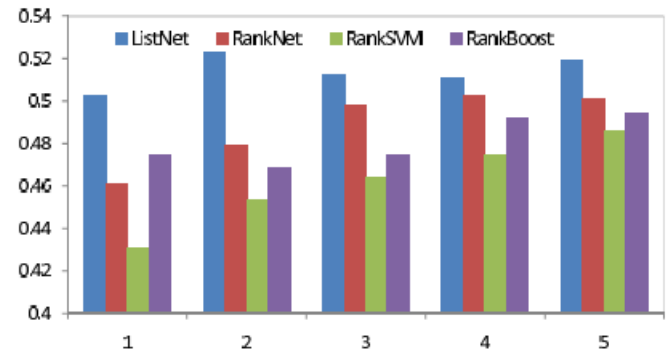

Figure 1. Ranking accuracies in terms of NDCG@n on TREC

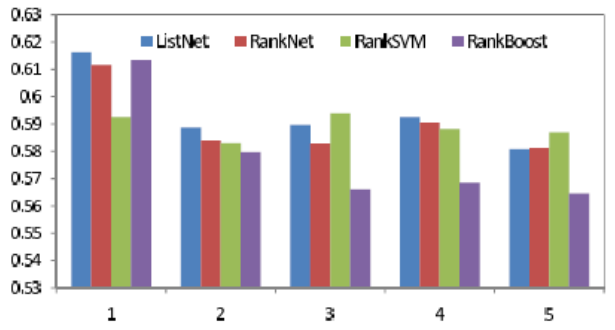

Figure 2. Ranking accuracies in terms of NDCG@n on OHSUMED

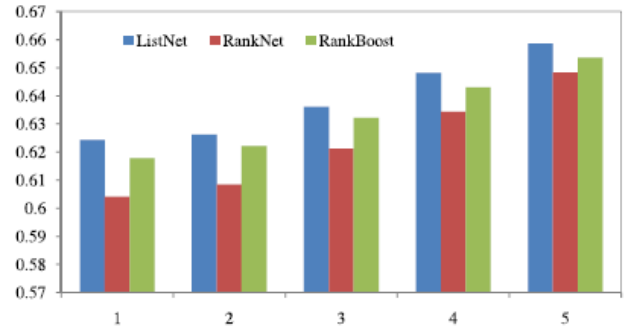

Figure 3. Ranking accuracies in terms of NDCG@n on CSearch

Experimental Results: Optimization processes

- Look at the ***correlations*** between the loss functions used by ListNet and RankNet and the measure of NDCG ***during the learning phase***.
- the pairwise loss of RankNet does not inversely correlate with NDCG most of the time.
- the listwise loss of ListNet ***completely inversely correlates*** with NDCG.
- pairwise loss converges more slowly than listwise loss,
  - *which means RankNet needs run more iterations in training than ListNet.*

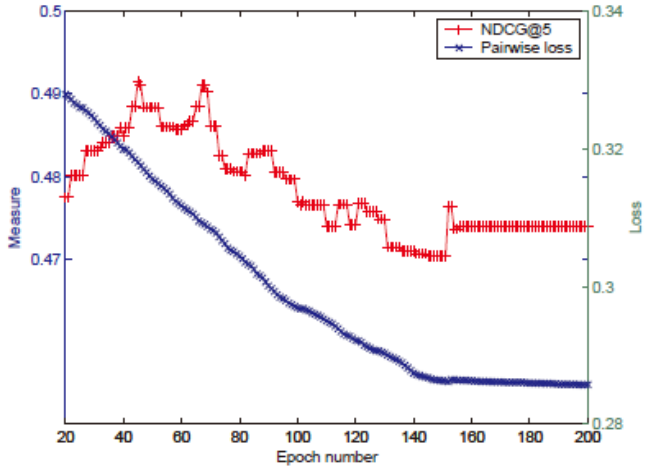

Figure 4. Pairwise loss v.s. NDCG@5 in RankNet

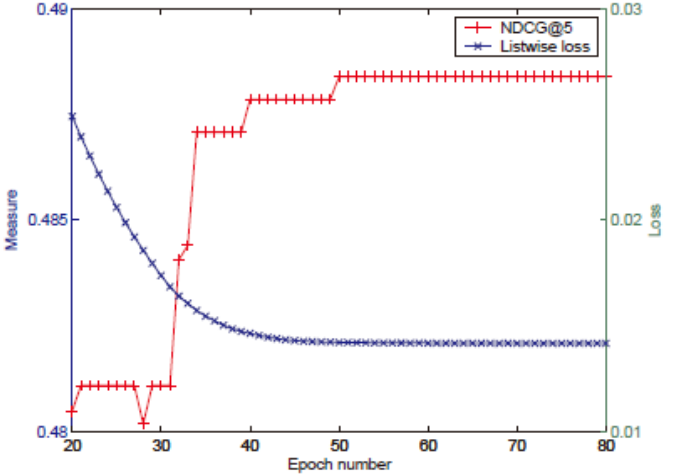

Figure 5. Listwise loss v.s. NDCG@5 in ListNet

Conclusions

- it is better to take this listwise approach than the traditional pairwise approach in learning to rank, using list of objects as instances instead of pairs in learning.
- the key issue for the listwise approach is to define a listwise loss function
  - *make use of probability models: permutation probability and top k probability*
  - *transform ranking scores into probability distributions.*
  - *utilize any metric between probability distributions*
- a learning method based on the approach,using Neural Network and Gradient Descent
- Experimental results show that the method works better than the existing pairwise methods
  - *suggesting that it is better to take the listwise approach in learning to rank*