Information Retrieval

Learning to Rank

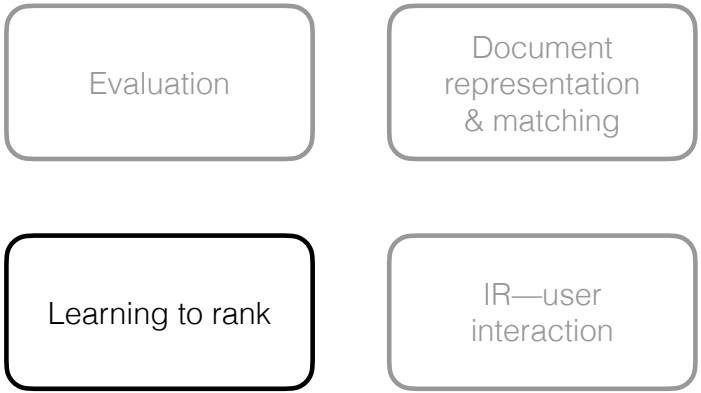
Harrie Oosterhuis

Radboud University
harrie.oosterhuis@ru.nl, https://harrieo.github.io

Partly based on the SIGIR 2017 Tutorial:
Neural Networks for Information Retrieval
Tom Kenter, Alexey Borisov, Christophe Van Gysel, Mostafa Dehghani, Maarten de Rijke, and Bhaskar Mitra.

Abridged by Ilya Markov

Learning to rank (LTR)



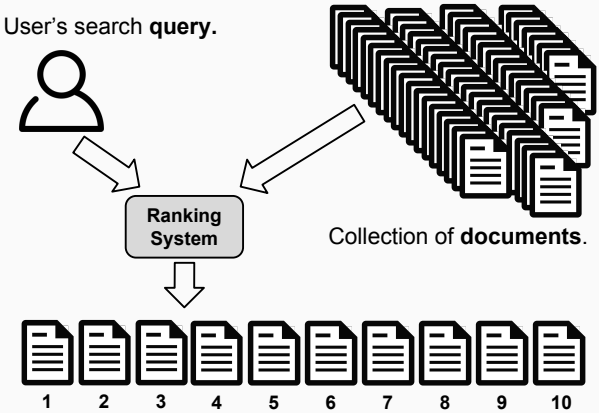
Introduction

Learning to rank (LTR)

Definition

"... the task to automatically construct a ranking model using training data, such that the model can sort new objects according to their degrees of relevance, preference, or importance." – Liu [2009]

Learning to rank (LTR)



Signals in Ranking Systems

Relevance signals:

1. Document-Query overlap
2. TF-IDF
3. BM25
4. Language Modelling
5. Neural Matching
6. ...

Other signals:

1. Pagerank (Do other pages link to this one.)
2. Webpage popularity
3. Spam detection (e.g. clickbait)
4. Query information (type, topic, etc)
5. Language of user/page/query
6. ...

Signals in Ranking Systems

Web search engines use a lot of these signals:

1. **Bing:** 136+ signals/features
2. **Istella:** 220+ signals/features
3. **Yahoo:** 700+ signals/features

How do we combine all these signals to create rankings?

With **machine learning**.

LTR: Preliminaries & Goal

Problem Formulation

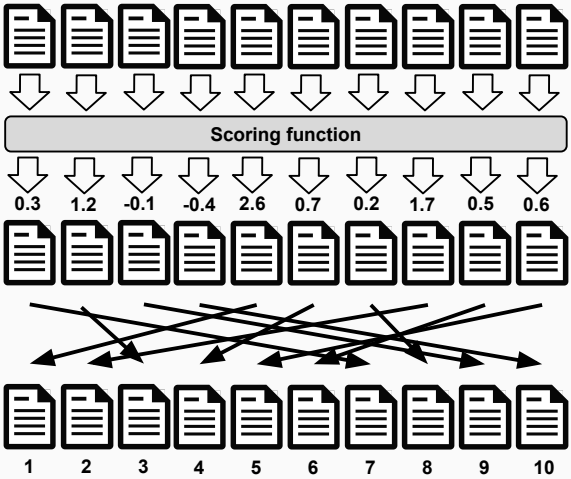
Representation:

- Represent the document and query in a format that a ML model can use: **a numerical vector** $\vec{x} \in \mathbb{R}^n$.

Prediction:

- Then a **ranking model** $f : \vec{x} \rightarrow \mathbb{R}$ is optimized to score each document-query combination so that relevant documents are scored higher.
- In mathematical terms: f maps a vector to a real-valued score.

Problem Formulation: Illustration



Features

We have already listed a lot of possible features that can be used.

Traditionally features are hand-crafted to encode IR insights,
nowadays we also have *deep learned* features.

They can be categorized as:

- **Document-only** or *static* features (e.g., document length)
- **Document-Query-combination** or *dynamic* features (e.g., BM25)
- **Query-only** features (e.g., query length)

Learning to rank data

Models can be trained on different data:

- **Offline or Supervised** LTR: learn from annotated data.
  - Expensive and time-consuming.
  - Provides ground-truth.
- **Online/Counterfactual** LTR: learn from user interactions.
  - Virtually free and easy to obtain.
  - Hard to interpret.

Supervised Data

This lecture is on **offline** LTR, we'll assume that we are rich.

Data is then obtained by:

1. Pay some humans to be annotators.
2. (Train them to be good annotators.)
3. Collect a set of queries.
4. Preselect a large (but not too large) set of documents per query.
5. Show document-query pairs to annotators.
6. Annotators rate every document-query pair on their relevance, (e.g. on a scale from 0 to 4).

Learning to Rank: Goal

Thus we have:

- Feature representation of document-query pairs: $\vec{x}_{q,d} \in \mathbb{R}$.
- Labels indicating the relevance of document-query pairs: $y_{q,d} \in [0, 4]$.

And we want:

- A function $f : \vec{x} \to \mathbb{R}$ that scores documents.
- To get the best ranking by sorting according to $f(\vec{x})$.

How do we find f?

The Pointwise Approach

Pointwise objectives

Regression-based or classification-based approaches are popular.

Regression loss

Given $\langle q, d \rangle$ predict *the value of* $y_{q,d}$.

E.g., *square loss* for binary or categorical labels,

$$\mathcal{L}_{Squared}(q, d, y_{q,d}) = \|y_{q,d} - f(\vec{x}_{q,d})\|^2, \tag{1}$$

where $y_{q,d}$ is the one-hot representation [Fuhr, 1989] or the actual value [Cossock and Zhang, 2006] of the label.

Pointwise objectives

Regression-based or classification-based approaches are popular.

Classification loss

Given $\langle q, d \rangle$ predict *the class* $y_{q,d}$.

E.g., *Cross-Entropy with Softmax* over categorical labels Y [Li et al., 2008],

$$\mathcal{L}_{\text{CE}}(q, d, y_{q,d}) = -log\Big(p(y_{q,d}|q, d)\Big) = -log\Big(\frac{e^{\gamma \cdot s_{y_{q,d}}}}{\sum_{y \in Y} e^{\gamma \cdot s_y}}\Big), \tag{2}$$

where $s_{y_{q,d}}$ is the model's score for label $y_{q,d}$.

Pointwise approaches to LTR

Regression loss

$$\mathcal{L}_{Squared} = \sum_{q,d} \|y_{q,d} - f(\vec{x}_{q,d})\|^2 \tag{3}$$

Classification loss

$$\mathcal{L}_{\text{CE}} = \sum_{q,d} -log\Big(p(y_{q,d}|q,d)\Big) = \sum_{q,d} -log\Big(\frac{e^{\gamma \cdot s_{y_{q,d}}}}{\sum_{y \in Y} e^{\gamma \cdot s_y}}\Big) \tag{4}$$

where $s_{y_{q,d}}$ is the model's score for label $y_{q,d}$.

What are **issues** with these approaches?

Minor issues with pointwise approaches

Some minor issues are:

- Class imbalance:
  - many irrelevant documents and very few relevant documents.
- Query level feature normalization is needed:
  - the distribution of features differs greatly per query.

These can be overcome.

Pointwise approaches to LTR

Regression loss

$$\mathcal{L}_{Squared} = \sum_{q,d} \|y_{q,d} - f(\vec{x}_{q,d})\|^2 \tag{5}$$

Classification loss

$$\mathcal{L}_{\text{CE}} = \sum_{q,d} -log\Big(p(y_{q,d}|q,d)\Big) = \sum_{q,d} -log\Big(\frac{e^{\gamma \cdot s_{y_{q,d}}}}{\sum_{y \in Y} e^{\gamma \cdot s_y}}\Big) \tag{6}$$

where $s_{y_{q,d}}$ is the model's score for label $y_{q,d}$.

What is **fundamentally wrong** with these methods?

The problem with pointwise LTR

Ranking is not a regression or classification problem.

A document-level loss does not work for ranking problems because document scores should not be considered independently.

In other words, pointwise methods **do not directly optimize ranking quality**.

The problem with pointwise LTR illustrated

What is the loss here?

$$\mathcal{L}_{Squared} = \sum_{q,d} \|y_{q,d} - f(\vec{x}_{q,d})\|^2 \tag{7}$$

The problem with pointwise LTR illustrated

What is the loss here?

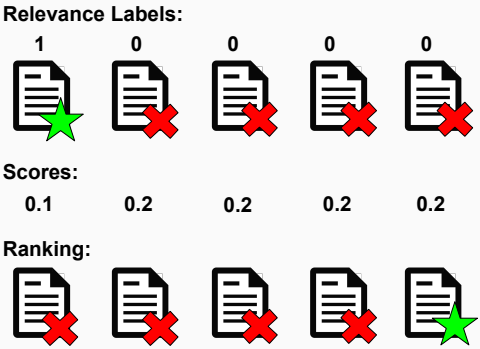
$$\mathcal{L}_{Squared} = 1.16 \tag{8}$$

The problem with pointwise LTR illustrated

What is the loss here?

$$\mathcal{L}_{Squared} = \sum_{q,d} \|y_{q,d} - f(\vec{x}_{q,d})\|^2 \tag{9}$$

The problem with pointwise LTR illustrated



What is the loss here?

$$\mathcal{L}_{Squared} = 0.97 \tag{10}$$

The problem with pointwise LTR illustrated

What is the loss here?

$$\mathcal{L}_{Squared} = 1.16 \tag{11}$$

Solution to pointwise?

Ranking is not a regression or classification problem.

A document-level loss does not work for ranking problems because document scores should not be considered independently.

In other words, pointwise method **do not directly optimize ranking quality**;

a lower loss does not mean a better ranking!

How can we solve this problem?

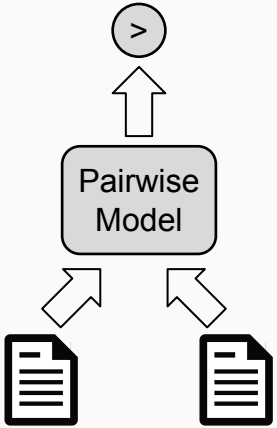
The Pairwise Approach

Pairwise objectives

Instead of looking at document-level, consider pairs of documents.

$$y_{q,d_i} > y_{q,d_j} \rightarrow d_i \succ d_j \tag{12}$$

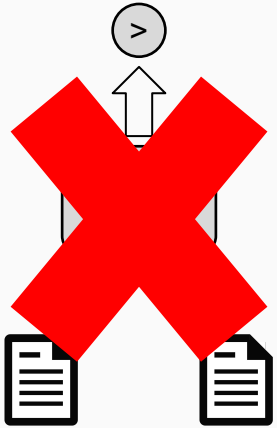
Naive Pairwise Model



Pairwise Model

$$P(d_i \succ d_j) = f(\vec{x}_i, \vec{x}_j) \tag{13}$$

Naive Pairwise Model



$$P(d_i \succ d_j) = f(\vec{x}_i, \vec{x}_j) \tag{14}$$

Naive Pairwise Model

Do **not** change the model to take **document pairs as input!**

$$P(d_i \succ d_j) = f(\vec{x}_i, \vec{x}_j) \tag{15}$$

This method would be quadratic in complexity: $O(N^2)$, during **inference**.

Pair-preferences have to be aggregated somehow.
This can lead to paradoxical situations:

$$\begin{aligned} d_1 &\succ d_2 \\ d_2 &\succ d_3 \\ d_3 &\succ d_1 \end{aligned} \tag{16}$$

The Pairwise Approach

The scoring model remains **unchanged**:

$$f(\vec{x}_i) = s_i \tag{17}$$

But the loss function is based on document pairs:

$$\mathcal{L}_{pairwise} = \sum_{d_i \succ d_j} \phi(s_i - s_j) \tag{18}$$

Thus we still score documents and then order according to scores.

Pairwise Loss Functions

Pairwise objectives

Pairwise loss minimizes the **average number of inversions** in ranking:

- $d_i \succ_q d_j$ but d_j is ranked higher than d_i

Pairwise loss generally has the following form [Chen et al., 2009],

$$\mathcal{L}_{pairwise} = \phi(s_i - s_j) \tag{19}$$

where ϕ can be

- Hinge function [Herbrich et al., 2000]: $\phi(z) = \max(0, 1 - z)$
- Exponential function [Freund et al., 2003]: $\phi(z) = e^{-z}$
- Logistic function [Burges et al., 2005]: $\phi(z) = \log(1 + e^{-z})$
- etc.

RankNet

RankNet

RankNet [Burges et al., 2005] is a *pairwise* loss function—popular choice for training neural LTR models and also an industry favourite [Burges, 2015].

Predicted probabilities: $P_{ij} = P(s_i > s_j) \equiv \frac{e^{\gamma \cdot s_i}}{e^{\gamma \cdot s_i} + e^{\gamma \cdot s_j}} = \frac{1}{1+e^{-\gamma(s_i - s_j)}}$

and $P_{ji} \equiv \frac{1}{1+e^{-\gamma(s_j - s_i)}}$

Desired probabilities: $\bar{P}_{ij} = 1$ and $\bar{P}_{ji} = 0$

Computing cross-entropy between $\bar{P}$ and P,

$$\begin{aligned} \mathcal{L}_{RankNet} &= -\bar{P}_{ij} \log(P_{ij}) - \bar{P}_{ji} \log(P_{ji}) \\ &= -\log(P_{ij}) \\ &= log(1 + e^{-\gamma(s_i - s_j)}) \end{aligned} \tag{20}$$

Deep Dive into RankNet

There is a famous factorization of RankNet [Burges, 2015, Burges et al., 2005].

Let $S_{ij} \in \{-1, 0, 1\}$ indicate the preference between d_i and d_j.
Then the desired probability for a pair is:

$$\bar{P}(d_i \succ d_j) = \frac{1}{2}(1 + S_{ij}). \tag{21}$$

The predicted probability is:

$$P(d_i \succ d_j) = \frac{1}{1 + e^{-\gamma(s_i - s_j)}}. \tag{22}$$

The cross-entropy loss is then:

$$\mathcal{L}_{ij} = \frac{1}{2}(1 - S_{ij})\gamma(s_i - s_j) + \log(1 + e^{-\gamma(s_i - s_j)}). \tag{23}$$

The cross-entropy loss is then:

$$\mathcal{L}_{ij} = \frac{1}{2}(1 - S_{ij})\gamma(s_i - s_j) + \log(1 + e^{-\gamma(s_i - s_j)}). \tag{24}$$

The derivate w.r.t. s_i:

$$\frac{\delta \mathcal{L}_{ij}}{\delta s_i} = \gamma \left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + e^{-\gamma(s_i - s_j)}} \right) = -\frac{\delta \mathcal{L}_{ij}}{\delta s_j}. \tag{25}$$

Then we can factorize the loss it so that:

$$\frac{\delta \mathcal{L}_{ij}}{\delta w} = \frac{\delta \mathcal{L}_{ij}}{\delta s_i}\frac{\delta s_i}{\delta w} + \frac{\delta \mathcal{L}_{ij}}{\delta s_j}\frac{\delta s_j}{\delta w} = \gamma \left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + e^{-\gamma(s_i - s_j)}} \right) \left(\frac{\delta s_i}{\delta w} - \frac{\delta s_j}{\delta w} \right). \tag{26}$$

The factorized cross entropy loss:

$$\frac{\delta \mathcal{L}_{ij}}{\delta w} = \gamma \left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + e^{-\gamma(s_i - s_j)}} \right) \left(\frac{\delta s_i}{\delta w} - \frac{\delta s_j}{\delta w} \right). \tag{27}$$

We choose λ so that:

$$\frac{\delta \mathcal{L}_{ij}}{\delta w} = \lambda_{ij} \left(\frac{\delta s_i}{\delta w} - \frac{\delta s_j}{\delta w} \right), \tag{28}$$

where:

$$\lambda_{ij} = \gamma \left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + e^{-\gamma(s_i - s_j)}} \right). \tag{29}$$

These lambdas act like *forces* pushing pairs of documents apart or together.

$$\lambda_{ij} = \gamma \left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + e^{-\gamma(s_i - s_j)}} \right). \tag{30}$$

These lambdas act like *forces* pushing pairs of documents apart or together.

On document level the same can be done:

$$\lambda_i = \sum_j \lambda_{ij} \tag{31}$$

The Pairwise Approach

The scoring model scores documents independently: $f(\vec{x}_{d_i}) = s_i$.

The loss is based on document pairs, and minimizes the number of incorrect inversions:

$$\mathcal{L}_{pairwise} = \sum_{d_i \succ d_j} \phi(s_i - s_j) \tag{32}$$

For instance, RankNet:

$$\mathcal{L}_{RankNet} = \sum_{d_i \succ d_j} log(1 + e^{-\gamma(s_i - s_j)}) \tag{33}$$

What is **wrong** with this approach?

The Pairwise Approach

Minor issue: RankNet is based on virtual probabilities: $P(d_i \succ d_j)$.

In reality the ranking model does not follow these probabilities.

Not elegant, but not a big deal.

The Pairwise Approach

The scoring model scores documents independently: $f(\vec{x}_{d_i}) = s_i$.

The loss is based on document pairs, and minimizes the number of incorrect inversions:

$$\mathcal{L}_{pairwise} = \sum_{d_i \succ d_j} \phi(s_i - s_j) \tag{34}$$

For instance, RankNet:

$$\mathcal{L}_{RankNet} = \sum_{d_i \succ d_j} log(1 + e^{-\gamma(s_i - s_j)}) \tag{35}$$

What is **fundamentally wrong** with this approach?

Problem with the Pairwise Approach

Ranking 1:

Ranking 2:

Which ranking do you think is better?

How many inversions does each ranking get correct?

Problem with the Pairwise Approach

Ranking 1:

Pairs correct: 9

Ranking 2:

Pairs correct: 10

The bottom ranking is better than the top according to the pairwise approach!

The Pairwise Approach

The scoring model scores documents independently: $f(\vec{x}_{d_i}) = s_i$.
The loss is based on document pairs, and minimizes the number of incorrect inversions:

$$\mathcal{L}_{pairwise} = \sum_{d_i \succ d_j} \phi(s_i - s_j) \tag{36}$$

However, **not every document pair is equally important**.

It is much **more important to get the correct ordering of top documents** than of the bottom documents.

For instance, the order of the top-5 is much more important than the order of documents after position 10.

The Listwise Approach

Holy grail of LTR

The **fundamental problem** with the approaches so far is that they did not optimize **ranking quality** directly.

A LTR method should directly optimize the ranking metric we care about.

What ranking metrics do we care about?

Ranking metrics can range from simple:

$$precision(R) = \frac{1}{|R|} \sum_{R_i} relevance(R_i) \tag{37}$$

to much more complex, e.g. discounted cumulative gain:

$$DCG(R) = \sum_{R_i} \frac{2^{relevance(R_i)} - 1}{\log(i+1)} \tag{38}$$

Evaluation Metrics in IR

How do we optimize for these metrics?

$$precision(R) = \frac{1}{|R|} \sum_{R_i} relevance(R_i) \tag{39}$$

$$\frac{\delta}{\delta w} precision(R) = \quad ??? \tag{40}$$

for discounted cumulative gain:

$$DCG(R) = \sum_{R_i} \frac{2^{relevance(R_i)-1}}{\log(i+1)} \tag{41}$$

$$\frac{\delta}{\delta w} DCG(R) = \quad ??? \tag{42}$$

These metrics are **non-continuous** and **non-differentiable**.

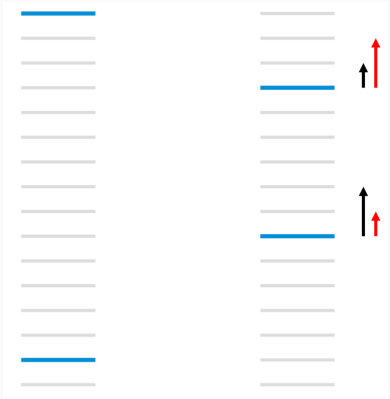
Listwise

Blue: relevant Gray: non-relevant

NDCG and ERR higher for left but pairwise errors less for right

Due to strong position-based discounting in IR measures, errors at higer ranks are much more problematic than at lower ranks

But listwise metrics are non-continuous and non-differentiable



[Burges, 2010]

LambdaRank

LambdaRank

Key observations:

- To train a model we do not need the costs themselves, only the gradients (of the costs w.r.t. model scores).
- The gradient should be bigger for pairs of documents that produces a bigger impact in NDCG by swapping positions.

LambdaRank [Burges et al., 2006] Multiply actual gradients with the change in NDCG by swapping the rank positions of the two documents:

$$\lambda_{LambdaRank} = \lambda_{RankNet} \cdot |\Delta \text{NDCG}| \tag{43}$$

LambdaRank

Multiply actual gradients with the change in NDCG by swapping the rank positions of the two documents:

$$\lambda_{LambdaRank} = \lambda_{RankNet} \cdot |\Delta \text{NDCG}|. \tag{44}$$

This approach also works with other metrics, e.g., $|\Delta \text{Precision}|$.

Empirically LambdaRank was shown to directly optimize IR metrics.

Recently, it was **theoretically proven** that LambdaRank optimizes a **lower bound** on certain IR metrics [Wang et al., 2018].

ListNet and ListMLE

ListNet and ListMLE

Create a probabilistic model for ranking, which is differentiable.

Sample documents from a Plackett-Luce distribution:

$$P(d_i) = \frac{\phi(s_i)}{\sum_{d_j \in D} \phi(s_j)} \tag{45}$$

For instance, $\phi(s_i) = e^{s_i}$:

$$P(d_i) = \frac{e^{s_i}}{\sum_{d_j \in D} e^{s_j}} \tag{46}$$

According to the Luce model [Luce, 2005], given four items $\{d_1, d_2, d_3, d_4\}$ the probability of observing a particular rank-order, say $[d_2, d_1, d_4, d_3]$, is given by:

$$P(\pi|s) = \frac{\phi(s_2)}{\phi(s_1) + \phi(s_2) + \phi(s_3) + \phi(s_4)} \cdot \frac{\phi(s_1)}{\phi(s_1) + \phi(s_3) + \phi(s_4)} \cdot \frac{\phi(s_4)}{\phi(s_3) + \phi(s_4)} \tag{47}$$

where π is a particular permutation and ϕ is a transformation (e.g., linear, exponential, or sigmoid) over the score s_i corresponding to item d_i.

ListNet and ListMLE

ListNet [Cao et al., 2007]

Compute the probability distribution over all possible permutations based on model score and ground-truth labels. The loss is then given by the KL-divergence between these two distributions.

This is computationally very costly, computing permutations of only the top-K items makes it slightly less prohibitive.

ListMLE [Xia et al., 2008]

Compute the probability of the ideal permutation based on the ground truth. However, with categorical labels more than one permutation is possible which makes this difficult.

Conclusion

Learning to Rank Quick Recap

Learning to Rank

Ranking is very important in places were **search or recommendation** is involved.
Methods should **scale** to large collections and work **fast** enough to help users.
Search engines use large numbers of signals/features.

Learning to Rank Quick Recap

- **Pointwise Approach**
  - Predict the **relevance per item**, simple but very naive.
  - **Ignores** that **ordering** of items is what matters.
- **Pairwise Approach**
  - **Loss** based on **document pairs**, minimize the number of incorrect inversions.
  - Ignores that **not** all document pairs have the **same impact**.
  - Often used in the industry.
- **Listwise Approach**
  - Tries to **optimize** for **IR metrics**, but they are **not differentiable**.
  - **Approximations** by heuristics, bounding or probabilistic approaches to ranking.
  - Best approach out of the three.

Harrie's Personal Opinion on the Point/Pair/List-wise Distiction

I do not think the distinction between pairwise and listwise is helpful:

- The derivatives of listwise losses always reduce to weighted pairwise derivatives (LambdaRank is explicit, ListNet does this implicitly).
- The LambdaRank with the Average Relevant Rank metric is equivalent to the pairwise RankNet loss.
- In the field there are often categorization mistakes, e.g., calling LambdaRank a pairwise method.

Personally I prefer non-ranking losses (pointwise) vs. ranking losses (pairwise & listwise).

Conclusion

In this lecture we discussed:

- why Learning to Rank is a separate category in Machine Learning.
- the main Learning to Rank approaches:
  - Pointwise, Pairwise, and Listwise.
- the problems/value of each approach.
- the most important methods:
  - RankNet, LambdaRank, ListNet.

Toolkits for off-line learning to rank

RankLib : https://sourceforge.net/p/lemur/wiki/RankLib

shoelace : https://github.com/rjagerman/shoelace [Jagerman et al., 2017]

QuickRank : http://quickrank.isti.cnr.it [Capannini et al., 2016]

RankPy : https://bitbucket.org/tunystom/rankpy

pyltr : https://github.com/jma127/pyltr

jforests : https://github.com/yasserg/jforests [Ganjisaffar et al., 2011]

XGBoost : https://github.com/dmlc/xgboost [Chen and Guestrin, 2016]

SVMRank : https://www.cs.cornell.edu/people/tj/svm_light [Joachims, 2006]

sofia-ml : https://code.google.com/archive/p/sofia-ml [Sculley, 2009]

pysofia : https://pypi.python.org/pypi/pysofia

References i

C. Burges. Ranknet: A ranking retrospective, 2015. Accessed July 16, 2017.

C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96. ACM, 2005.

C. J. Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81, 2010.

C. J. Burges, R. Ragno, and Q. V. Le. Learning to rank with nonsmooth cost functions. In *NIPS*, volume 6, pages 193–200, 2006.

Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136. ACM, 2007.

G. Capannini, C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, and N. Tonellotto. Quality versus efficiency in document scoring with learning-to-rank models. *IPM*, 52(6):1161–1177, 2016.

References ii

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *KDD*, pages 785–794. ACM, 2016.

W. Chen, T.-Y. Liu, Y. Lan, Z.-M. Ma, and H. Li. Ranking measures and loss functions in learning to rank. In *Advances in Neural Information Processing Systems*, pages 315–323, 2009.

D. Cossock and T. Zhang. Subset ranking using regression. In *COLT*, volume 6, pages 605–619. Springer, 2006.

Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *Journal of machine learning research*, 4(Nov):933–969, 2003.

N. Fuhr. Optimum polynomial retrieval functions based on the probability ranking principle. *ACM Transactions on Information Systems (TOIS)*, 7(3):183–204, 1989.

Y. Ganjisaffar, R. Caruana, and C. Lopes. Bagging gradient-boosted trees for high precision, low variance ranking models. In *SIGIR*, pages 85–94. ACM, 2011.

R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. 2000.

References iii

R. Jagerman, J. Kiseleva, and M. de Rijke. Modeling label ambiguity for neural list-wise learning to rank. In *Neu-IR SIGIR Workshop*, 2017.

T. Joachims. Training linear svms in linear time. In *KDD*, pages 217–226. ACM, 2006.

P. Li, Q. Wu, and C. J. Burges. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Advances in neural information processing systems*, pages 897–904, 2008.

T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

R. D. Luce. *Individual choice behavior: A theoretical analysis*. Courier Corporation, 2005.

D. Sculley. Large scale learning to rank. In *In NIPS 2009 Workshop on Advances in Ranking*, 2009.

X. Wang, C. Li, N. Golbandi, M. Bendersky, and M. Najork. The lambdaloss framework for ranking metric optimization. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1313–1322, 2018.

F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*, pages 1192–1199. ACM, 2008.