Learning to Rank

Zoekmachines 2024 - IR0

Philipp Hager

September 24, 2024

University of Amsterdam
p.k.hager@uva.nl

About me

- PhD student with Maarten de Rijke
- IRLab at UvA and the Mercury ML Lab at Booking.com
- **Previously:**
  - Recommender systems at Blinkist, Berlin
  - M.Sc. Hasso-Plattner Institute, Potsdam
  - B.Sc. University of Applied Sciences, Düsseldorf
- **Research interests:** Ranking with user interactions, user simulation, and evaluating IR systems offline.

Motivation

Google

attractions in Amsterdam

TripAdvisor

https://www.tripadvisor.com › Attractions-g188590-A...

THE 15 BEST Things to Do in Amsterdam

Top **Attractions in Amsterdam** · 1. Anne Frank House · 2. Van Gogh Museum · 3. Rijksmuseum · 4. Vondelpark · 5. The Jordaan · 6. Centraal Station · 7. Heineken ...

Attractions: 3,115 Attraction Photos: 309,790

Attraction Reviews: 645,083

iamsterdam.com

https://www.iamsterdam.com › see-and-do › attraction...

Attractions and sights | I amsterdam

Most popular **attractions** · Heineken Experience · ARTIS · Koninklijk Paleis (Royal Palace) · Anne Frank House · **Amsterdam** Canal Cruise - 100 highlights · Johan Cruijff ...

Free things to do in Amsterdam · Rembrandts Amsterdam... · This is holland

PlanetWare

https://www.planetware.com › amsterdam-nl-nh-amst

24 Top-Rated Tourist Attractions in Amsterdam

Signals in Web Search

Textual Signals:

- **Query content**: text
- **Document content**: title, page content

How well does the query text match the document text? [6]

- BM-25
- TF-IDF / vector space models
- Semantic matching with LLMs or topic models

Signals in Web Search

But there are many signals beyond text:

- **Query**: type, language
- **Document**: urls, images, structure
- **User context**: location, date, device, search history
- **Metadata**: popularity, recency, page quality, spam, adult content, ...
- **External stakeholders**: advertisers, auctions, content creators, ...

Signals in Web Search

Modern search engines use a lot of features:

- **Airbnb** [10]: > 195 features
- **Bing** [15]: > 136 features
- **Istella** [7]: > 220 features
- **Yahoo** [2]: > 700 features

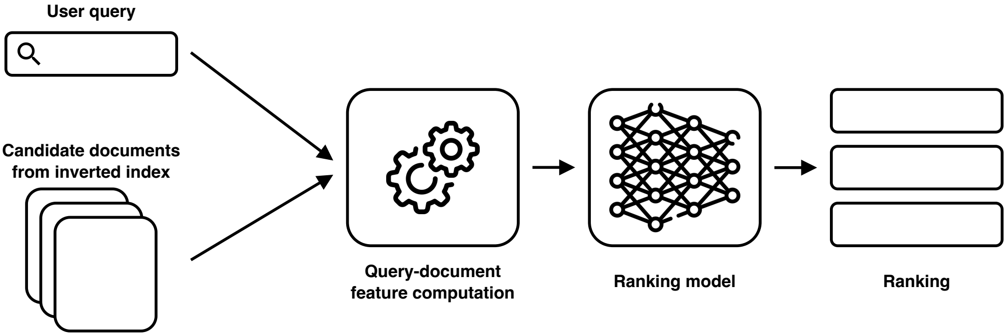
How do we combine all of these signals?

Learning to Rank

Learning to Rank (LTR) is

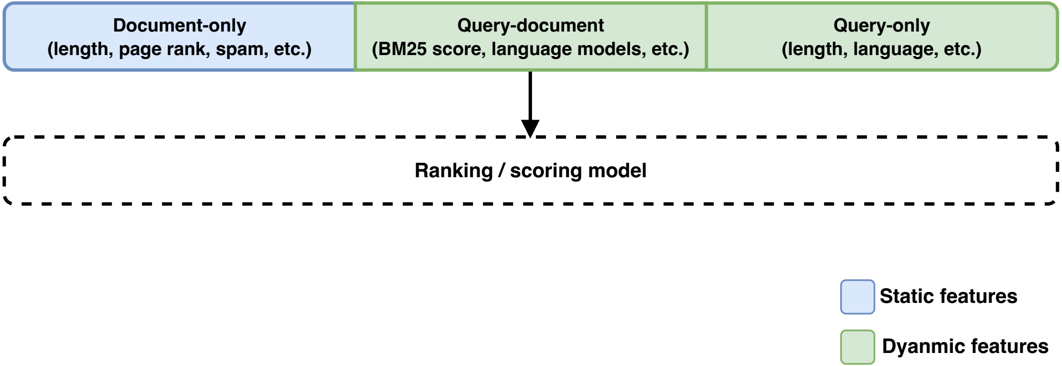
"... a task to automatically construct a ranking model using training data, such that the model can sort new objects according to their degrees of relevance, preference, or importance." – Liu [13]

Learning to Rank

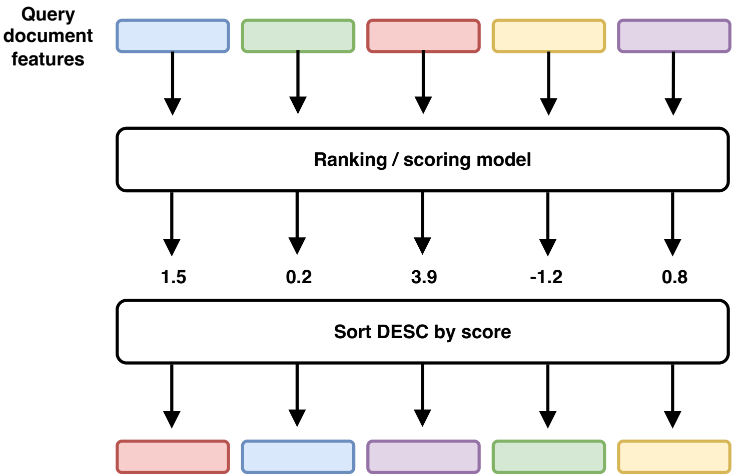


Features

Traditionally LTR used *hand-crafted* numerical features,
nowadays, we also use *deep learned* features. We can categorize features into:



Problem Formulation



The goal of learning to rank

Thus, we have:

- a **feature vector** for each query-document pair: $\vec{x}_{q,d} \in \mathbb{R}^m$
- a **relevance judgment** for each query-document pair, e.g.: $y_{q,d} \in \{0, 1, 2, 3, 4\}$

A ranking model $f : \vec{x} \rightarrow \mathbb{R}$ scores each query-document pair to optimize the order of items when sorting descendingly by $f(\vec{x}_{q,d}) = s_{q,d}$.

How can we learn a ranking model f?

Pointwise Methods

Pointwise LTR

Regression: Relevance as a real-valued score [4, 9]

For example, we can use linear regression for our ranking model:

$$f(x) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = s$$

We assign a **weight** w to each feature in x to minimize
the error between the predicted scores and true relevance.

Usually we quantify how far off our predictions are using the **mean squared error** loss:

$$\mathcal{L}_{mse} = \frac{1}{n} \sum_{i=1}^{n} (y_i - s_i)^2$$

Pointwise LTR - MSE

Relevance labels y	Predicted scores s	Squared Error
		$(y - s)^2$
1	0.9	$(1 - 0.9)^2 = 0.01$
0	0.7	$(0 - 0.7)^2 = 0.49$
1	0.6	$(1 - 0.6)^2 = 0.16$
0	0.2	$(0 - 0.2)^2 = 0.04$
0	0.1	$(0 - 0.1)^2 = 0.01$
		MSE = 0.142

Pointwise LTR

Regression: Relevance as a real-valued score [4, 9]

Classification: Relevance as *unordered* categories [3, 14]

Ordinal regression: Relevance as *ordered* categories [5, 16]

What are challenges with these approaches?

Problems of pointwise LTR

Some (solvable) challenges include:

- **Class imbalance**: We have way more irrelevant than relevant documents
- **Feature normalization**: Feature distributions can differ greatly between queries

Problems of pointwise LTR

A more fundamental problem:
Pointwise methods predict a score for each query-document independently,
but document scores are fundamentally dependent on each other in a ranking.

Meaning, minimizing a pointwise loss does not always lead to a better ranking.

Pointwise LTR: A lower loss does not imply a better ranking

Relevance labels	Predicted scores
1	0.6
0	0.5
0	0.5
0	0.5
0	0.5

What is the loss?

$$\mathcal{L}_{mse} = \frac{1}{n}\sum_{i=1}^{n}(y_i - s_i)^2$$

Pointwise LTR: A lower loss does not imply a better ranking

Relevance labels	Predicted scores
1	0.6
0	0.5
0	0.5
0	0.5
0	0.5

$$\text{Loss } \mathcal{L}_{mse} = 1.16$$
$$\text{MRR} = 1, \text{ nDCG} = 1$$

Pointwise LTR: A lower loss does not imply a better ranking

Relevance labels	Predicted scores
1	0.2
0	0.2
0	0.2
0	0.2
0	0.1

Loss $\mathcal{L}_{mse} = 0.97$

MRR $= 0.2$, nDCG $= 0.39$

Pairwise Methods

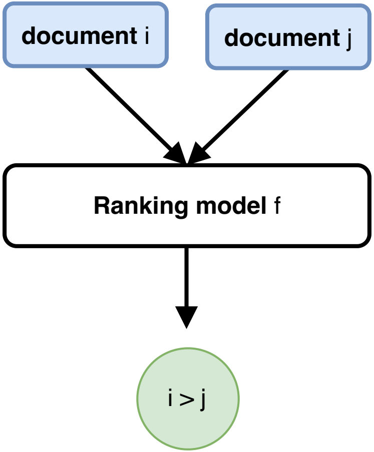
Pairwise LTR

Observation: For a good ranking, we only require relative relevance levels:

$$y_i > y_j \rightarrow s_i > s_j$$

How can we optimize a model with pairs of items?

Naive Pairwise Model



$$P(i > j) = f(x_i, x_j)$$

Naive Pairwise Model

Let's (naively) change the ranking model to take **document pairs as input**:

$$P(i > j) = f(x_i, x_j)$$

But pairwise document inputs are not a good idea:

- This method has quadratic complexity $O(N^2)$ during **training** and **inference** and thus does not scale to many documents.
- Pair-wise preferences have to be aggregated and can lead to paradoxical situations:

$$s_1 > s_2$$

$$s_2 > s_3$$

$$s_3 > s_1$$

Pairwise LTR

A better idea: Let's keep the ranking model **unchanged**:

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

But the **loss function** is based on pairs of documents:

$$\mathcal{L}_{pairwise}(s, y) = \sum_{y_i > y_j} \phi(s_i - s_j)$$

We still score one document at-a-time and can sort according to scores, but the model is optimized over item pairs.

Pairwise LTR

Pairwise loss functions minimize the number of incorrectly ranked pairs:

where $y_i > y_j$, but our model falsely predicts $s_i < s_j$.

Pairwise loss functions generally have the following form:

$$\mathcal{L}_{pairwise}(s, y) = \sum_{y_i > y_j} \phi(s_i - s_j)$$

where ϕ can be the:

- **Hinge** function in RankingSVM [11, 12]: $\phi(z) = \max(0, 1 - z)$
- **Exponential** function in RankBoost [8]: $\phi(z) = e^{-z}$
- **Logistic / Sigmoid** function in RankNet [1]: $\phi(z) = \log(1 + e^{-z})$

RankNet

RankNet

Introduced by Burges et al. [1] in 2005 to train neural ranking models.
Popular in industry applications and won the ICML 2015 test of time award.[1]

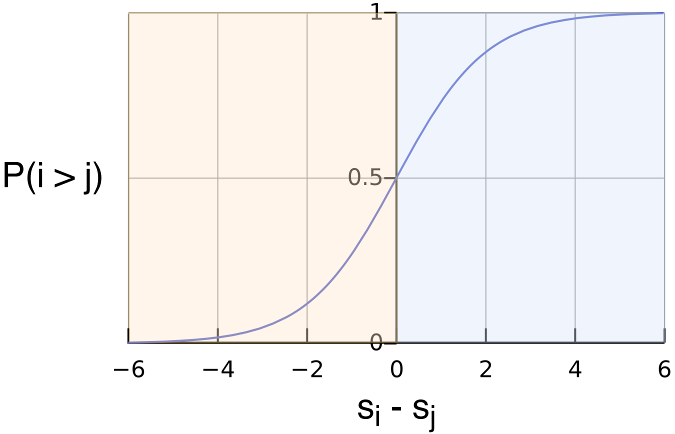
RankNet defines the probability that document i should be ranked over document j as:

$$P(i > j) = \text{sigmoid}(s_i - s_j)$$

RankNet then uses the log loss between the predicted probabilities for each pair and their true/target probability: $\bar{P}(i > j)$.

[1] `https://icml.cc/2015/index.html%3Fp=51.html`

RankNet



Mapping the difference in scores between to items to the **predicted probability** $P(i > j)$ using the sigmoid function.

RankNet

Now that we have a model prediction for each item pair,
where do we get the **target probability** of $\bar{P}(i > j)$ from?

1. Ask annotators to judge pairs of items (infeasible, it requires $O(N^2)$ annotations).
2. We make up probabilities based on relevance judgments:
   - If $y_i > y_j$, set $\bar{P}(i > j) = 1$
   - If $y_i = y_j$, set $\bar{P}(i > j) = 0.5$
   - If $y_i < y_j$, set $\bar{P}(i > j) = 0$

These *made-up/virtual* target probabilities were chosen mainly for convenience as they simplify the final loss to:

$$\mathcal{L}_{RankNet}(s, y) = \sum_{y_i > y_j} \log(1 + e^{-(s_i - s_j)})$$

What are problems of pairwise methods?

The made-up target probabilities $\bar{P} \in \{0, 0.5, 1\}$ are quite crude, since any difference in relevance labels is treated equally. For example:

$$P(4 > 1) = 1.0$$
$$P(4 > 3) = 1.0$$

Not very elegant, but works well in practice...

A more important limitation: We treat all item pairs as equally important, but are they?

Pairwise LTR: Minimizing pairwise errors

Pairwise LTR: Minimizing pairwise errors

Reducing pairwise errors from 13 (left) to 11 (right),
while top-heavy measures like MRR and nDCG degrade [1, Figure 1].

Pairwise LTR: Minimizing pairwise errors

The **black** arrows denote the RankNet gradients,
while what we'd arguably want are the **red** arrows [1, Figure 1].

Listwise Methods

LambdaRank

Motivation: Can we directly optimize IR metrics such as nDCG, Precision, and MRR?

Reciprocal Rank: Reciprocal of the rank of the first relevant item after sorting by our scores s:

$$\mathrm{RR} = \frac{1}{\mathrm{rank}_i}$$

Discounted Cumulative Gain:

$$\mathrm{DCG} = \frac{1}{n}\sum_{i=1}^{n} \frac{2^{y_i} - 1}{\log(i+1)}$$

Non-smooth and discontinuous

- Ranking metrics typically only **depend on the rank of an item, not on its score**
- Model **scores change smoothly**, the **ranks of documents change abruptly**

Listwise LTR

Non-differentiable

Ranking metrics rely on a sorting operation that is non-smooth and discontinuous w.r.t. to model parameters θ:

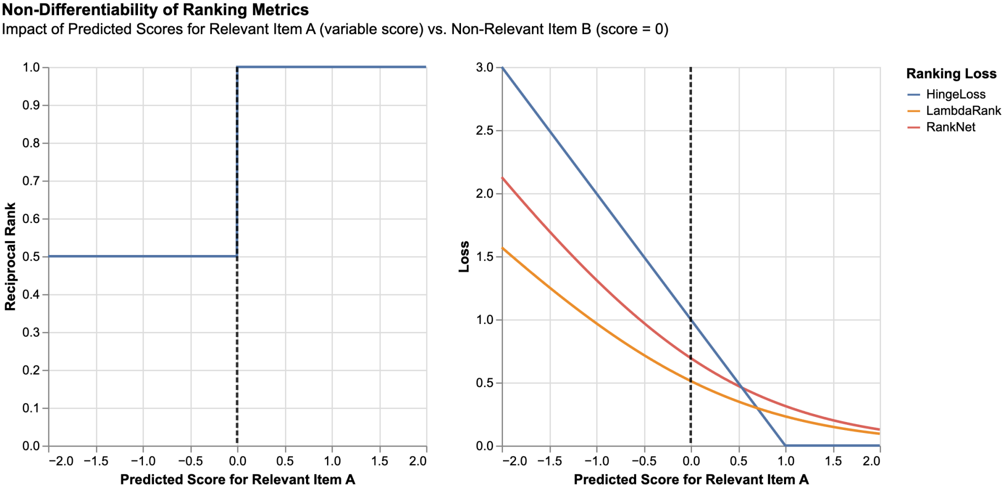
$$\frac{\partial \text{RR}}{\partial \theta} = ???$$

$$\frac{\partial \text{DCG}}{\partial \theta} = ???$$

Thus, ranking metrics are either **flat** (with zero gradient) or **discontinuous**

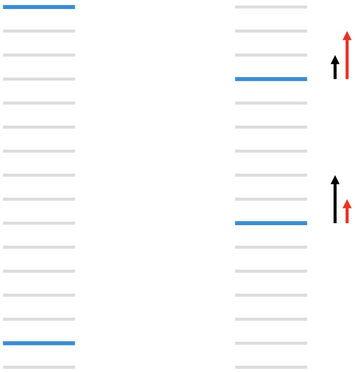
Holy grail of LTR: Finding methods that (indirectly) optimize listwise IR metrics

Listwise LTR



LambdaRank

LambdaRank



Observations:

I. To train a model, we don't need the costs just the gradients (of the costs w.r.t model scores)

II. Gradients should be larger for pairs that have a greater impact on our metric

LambdaRank

Idea: Scale the RankNet gradients for each document pair based on the change in nDCG we would observe after swapping the two items:

$$\mathcal{L}_{LambdaRank}(s, y) = \sum_{y_i > y_j} \Delta\text{NDCG}(i, j) \log\left(\text{sigmoid}(s_i - s_j)\right)$$

When implementing LambdaRank using multiple additive regression trees (MART), it's called **LambdaMART** and is a very strong baseline methods for learning to rank.

Conclusion

Summary

To recap:

- Today's search and recommender systems use many signals for ranking.
- These signals can be computed beforehand (static features) or have to be computed when the user submits their query (dynamic features).
- Learning to rank is a method to learn models that automatically combine these query and document features into a single ranking.

Summary

Pointwise

- Predict **relevance independently of other items**
- Ignores that the predicted scores are used to sort items

Pairwise

- **Loss** is based on **item pairs** and minimizes the number of incorrectly ranked pairs
- Ignores that **not all document pairs have the same impact** on ranking metrics

Listwise

- Optimize a list of items based on **non-differentiable ranking metrics**
- **Approximations** by heuristics, bounding, or probabilistic ranking methods
- Typically considered the strongest method out of the three

Summary

In this lecture, we discussed:

- the motivation behind **learning to rank**
- the **pointwise, pairwise, and listwise approaches** to LTR, their pros and cons
- the intuition behind the **most important algorithms**: RankNet and LambdaRank

And with that, thanks for listening. Are there any remaining questions?

References

[1] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 89–96, 2005. doi: 10.1145/1102351.1102363. URL https://doi.org/10.1145/1102351.1102363.

[2] Olivier Chapelle and Yi Chang. Yahoo! learning to rank challenge overview. In *Proceedings of the Learning to Rank Challenge*, volume 14 of *Proceedings of Machine Learning Research (PMLR)*, pages 1–24, 6 2011. URL https://proceedings.mlr.press/v14/chapelle11a.html.

[3] William S. Cooper, Fredric C. Gey, and Daniel P. Dabney. Probabilistic retrieval based on staged logistic regression. In *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 198–210, 1992. doi: 10.1145/133160.133199. URL https://doi.org/10.1145/133160.133199.

[4] David Cossock and Tong Zhang. Subset ranking using regression. In *Proceedings of the Annual Conference on Learning Theory (COLT)*, pages 605–619, 2006. doi: 10.1007/11776420_44. URL https://doi.org/10.1007/11776420_44.

[5] Koby Crammer and Yoram Singer. Pranking with ranking. In *Advances in Neural Information Processing Systems (NIPS)*, volume 14, 2001. URL https://proceedings.neurips.cc/paper_files/paper/2001/file/5531a5834816222280f20d1ef9e95f69-Paper.pdf.

[6] W Bruce Croft, Donald Metzler, and Trevor Strohman. *Search engines: Information retrieval in practice*, volume 520. Addison-Wesley Reading, 2010.

[7] Domenico Dato, Sean MacAvaney, Franco Maria Nardini, Raffaele Perego, and Nicola Tonellotto. The istella22 dataset: Bridging traditional and neural learning to rank evaluation. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 3099–3107, 2022. doi: 10.1145/3477495.3531740. URL https://doi.org/10.1145/3477495.3531740.

[8] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research (JMLR)*, 4:933–969, 2003. ISSN 1532-4435.

[9] Norbert Fuhr. Optimum polynomial retrieval functions based on the probability ranking principle. *ACM Transactions on Information Systems (TOIS)*, 7(3):183–204, 1989. ISSN 1046-8188. doi: 10.1145/65943.65944. URL https://doi.org/10.1145/65943.65944.

[10] Malay Haldar, Mustafa Abdool, Prashant Ramanathan, Tao Xu, Shulin Yang, Huizhong Duan, Qing Zhang, Nick Barrow-Williams, Bradley C. Turnbull, Brendan M. Collins, and Thomas Legrand. Applying deep learning to Airbnb search. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1927–1935, 2019. doi: 10.1145/3292500.3330658. URL https://doi.org/10.1145/3292500.3330658.

[11] Ralf Herbrich, Thore Graepel, and Klause Obermayer. Large margin rank boundaries for ordinal regression. In *Advances in Large Margin Classifiers*, chapter 7, pages 115–132. The MIT Press, 1999. URL http://www.herbrich.me/papers/nips98_ordinal.pdf.

[12] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 133–142, 2002. doi: 10.1145/775047.775067. URL https://doi.org/10.1145/775047.775067.

[13] Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009. doi: 10.1561/1500000016. URL https://doi.org/10.1561/1500000016.

[14] Ramesh Nallapati. Discriminative models for information retrieval. In *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 64–71, 2004. doi: 10.1145/1008992.1009006. URL https://doi.org/10.1145/1008992.1009006.

[15] Tao Qin and Tie-Yan Liu. Introducing LETOR 4.0 datasets. *CoRR*, abs/1306.2597, 2013. URL http://arxiv.org/abs/1306.2597.

[16] Amnon Shashua and Anat Levin. Ranking with large margin principle: Two approaches. In *Advances in Neural Information Processing Systems (NIPS)*, volume 15, 2002. URL https://proceedings.neurips.cc/paper_files/paper/2002/file/51de85ddd068f0bc787691d356176df9-Paper.pdf.