Master on Foundations of Data Science

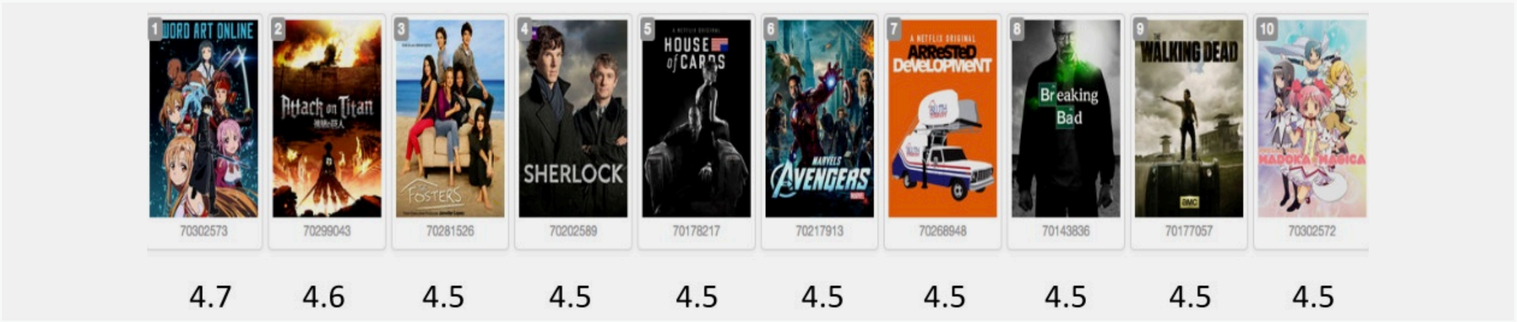
Recommender Systems

Learning to Rank

Santi Seguí | **2021-2022**

Ranking

- Most of the recommendations are **presented in a sorted list**.



- Recommendation can be understood as a ranking problem.

Optimized to achieve the best

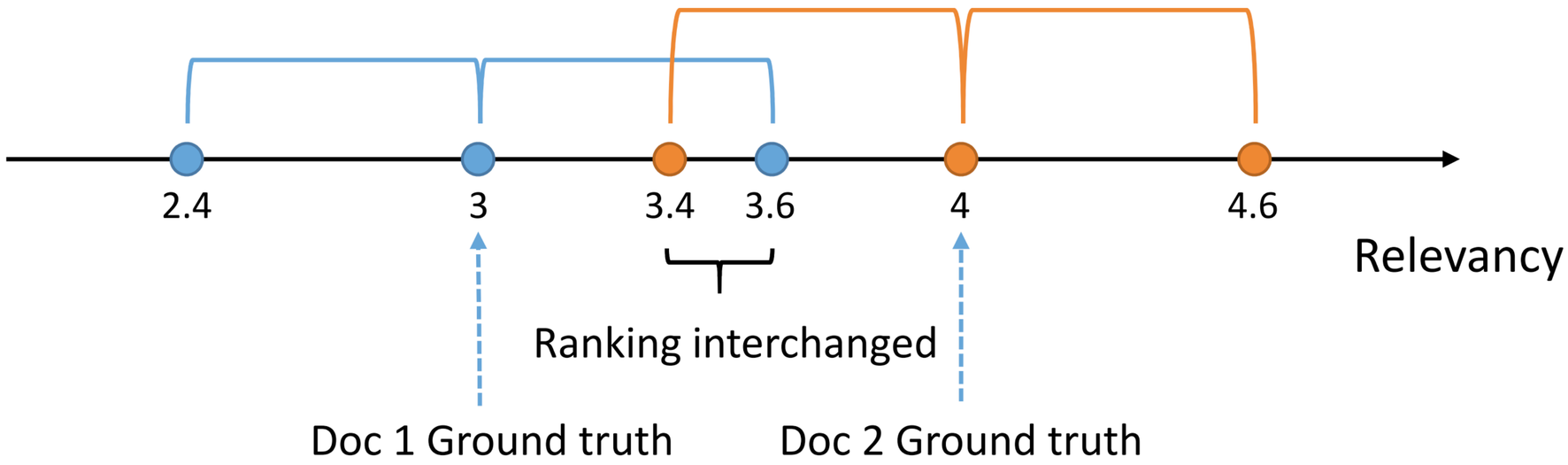
Accuracy

RMSE, MAE…

Accuracy

Any **problem** with RMSE or MAE?

Rating Prediction vs. Ranking

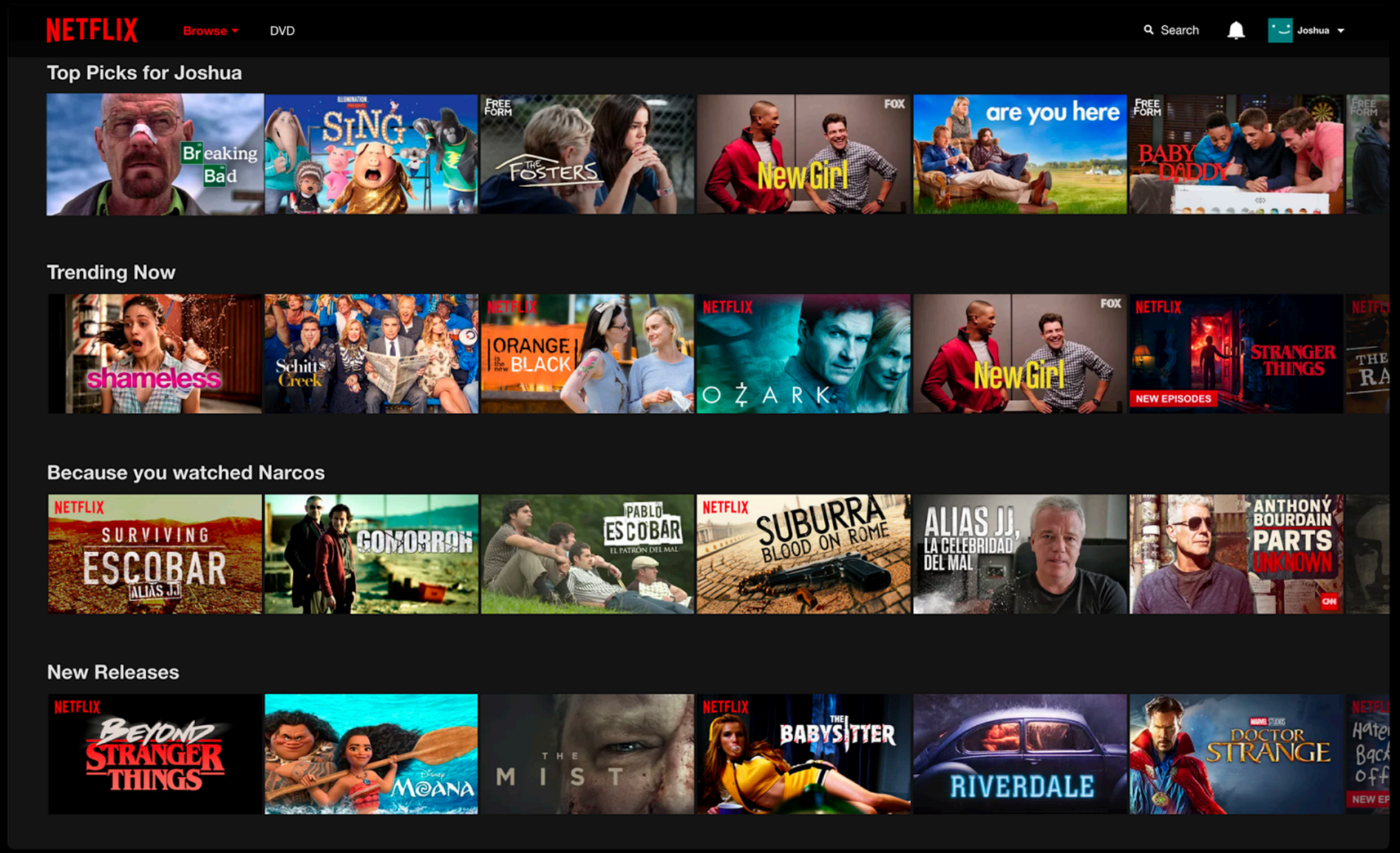


Same RMSE/MAE might lead to different rankings

Accuracy

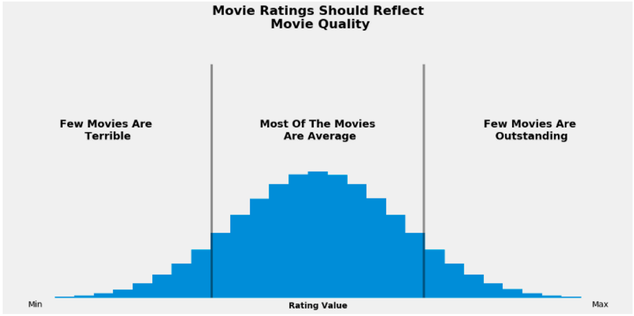
Any **problem** with RMSE or MAE?

these metrics do **not** really **measure the user experience**


NETFLIX
Browse
DVD
Search
Joshua
Top Picks for Joshua
Breaking Bad
SING
THE FOSTERS
New Girl
are you here
BABY DADDY
Trending Now
shameless
Schitt's Creek
ORANGE is the new BLACK
OZARK
New Girl
STRANGER THINGS
NEW EPISODES
Because you watched Narcos
SURVIVING ESCOBAR
ALIAS JJ
GOMORRON
PABLO ESCOBAR
EL PATRÓN DEL MAL
SUBURRA
BLOOD ON ROME
ALIAS JJ, LA CELEBRIDAD DEL MAL
ANTHONY BOURDAIN PARTS UNKNOWN
New Releases
BEYOND STRANGER THINGS
MOANA
THE MIST
THE BABYSITTER
RIVERDALE
DOCTOR STRANGE

Top Ranking

- If we serve the recommendation as a top-ranked list **RMSE**, **MAE** and other similar metrics are not appropriate to evaluate how good is the system.



Higher **accuracy** does **not** always mean higher **satisfaction**

Solution: Consider other behaviors

Being accurate is not enough: how accuracy metrics have hurt recommender systems
SM McNee, J Riedl, JA Konstan
CHI'06 extended abstracts on Human factors in computing systems, 1097-1101

680 2006

How to evaluate the ranking?

Rank **Evaluation**

Top-N Recommendations

- Populars measures to evaluate the **quality of top-N rankings** are the following:
  - Precision@K
  - Mean Average Precision (MAP)
  - Normalized Discounted cumulative gain (NDCG)

Rank **Evaluation**

- Recall@K

$$recall@K = \frac{\text{number of items that the user likes among the top K}}{\text{total number of items that the user likes}}$$

Rank **Evaluation**

- Precision@K

$$precision@K = \frac{\text{relevant items in the top k results}}{\text{k}}$$

Rank **Evaluation**

- **Mean Average Precision (MAP)** calculates the precision at the position of every corrected item in the ranked results list

$$AP = \frac{\sum_k P@k \cdot y_i(k)}{\text{number o frelevant documents}}$$

$$MAP = \frac{\sum_{q=1}^{Q} AP(q)}{Q}$$

Query 1:  AP=⅓(1/1+⅔+3/6)=0.72

Query 2:  AP=⅓(1/1+2/2+¾)=0.917

MAP= (0.72+0.917)/2=0.8185

Rank **Evaluation**

Case: Kaggle Challenge Expedia

Submissions are evaluated according to the Mean Average Precision @ 5 (MAP@5):

$$MAP@5 = \frac{1}{|U|}\sum_{u=1}^{|U|}\sum_{k=1}^{min(5,n)} P(k)$$

where |U| is the number of user events, P(k) is the precision at cutoff k, n is the number of predicted hotel clusters.

Rank **Evaluation**

- **Normalized Discounted cumulative gain (NDCG).**
  - If there is items which are more relevant to others (e.g. ratings)

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

graded relevance to get a stronger emphasis on retrieving relevant documents

$$\mathrm{IDCG_p} = \sum_{i=1}^{|REL_p|} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

list of relevant documents (ordered by their relevance) in the corpus up to position p

$$\mathrm{nDCG_p} = \frac{DCG_p}{IDCG_p}$$

Rank **Evaluation**

Cumulative Gain

$$\mathrm{CG_p} = \sum_{i=1}^{p} rel_i$$

p elements in the recommended ranking

graded relevance of the result at position i (gain)

Discounted Cumulative Gain (standard)

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)}$$

logarithmic reduction factor to penalize proportionally to the position of the result

Discounted Cumulative Gain (industry)

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

graded relevance to get a stronger emphasis on retrieving relevant documents

Ideal Discounted Cumulative Gain

$$\mathrm{IDCG_p} = \sum_{i=1}^{|REL_p|} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

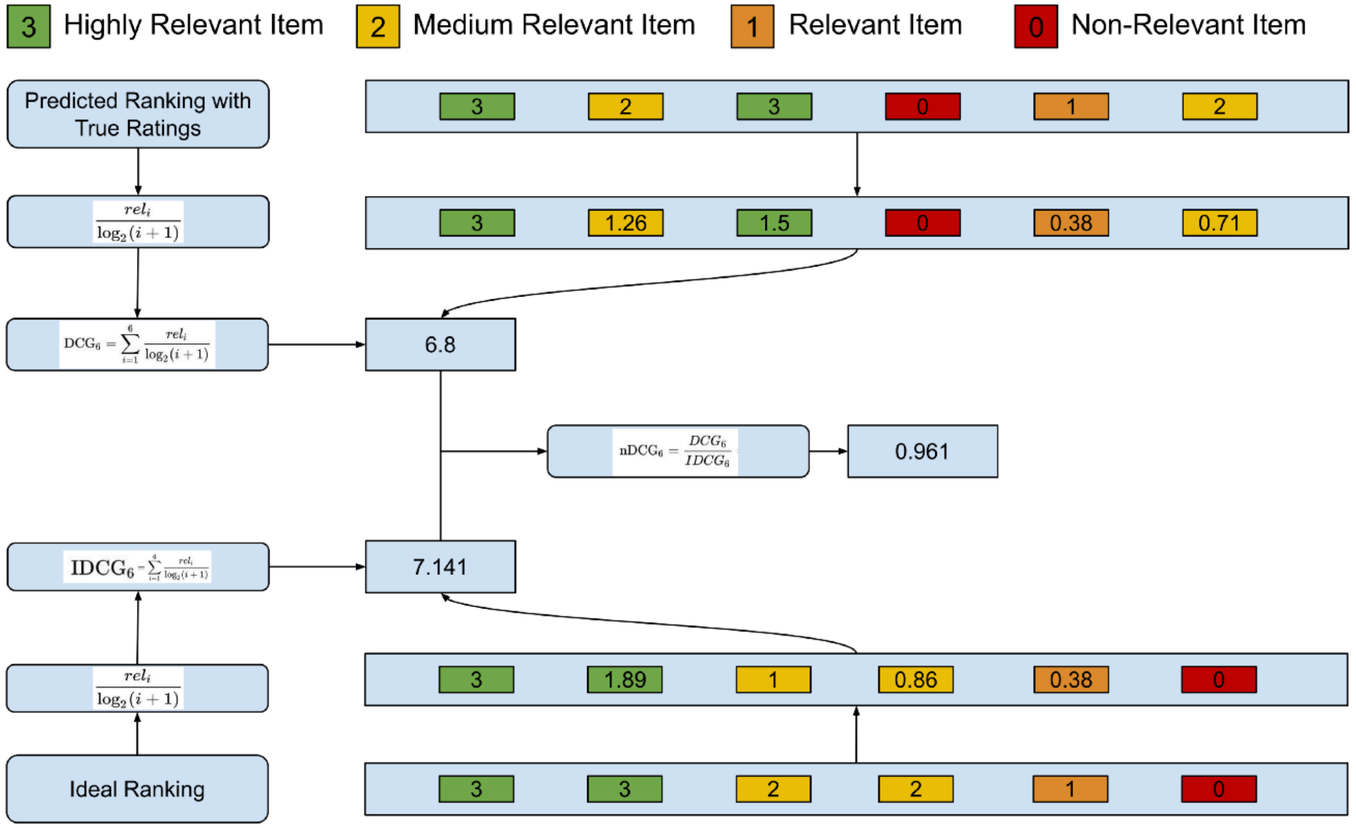
list of relevant documents (ordered by their relevance) in the corpus up to position p

Normalized Discounted Cumulative Gain

$$\mathrm{nDCG_p} = \frac{DCG_p}{IDCG_p}$$

Average Normalized Discounted Cumulative Gain Across Users

Rank **Evaluation**

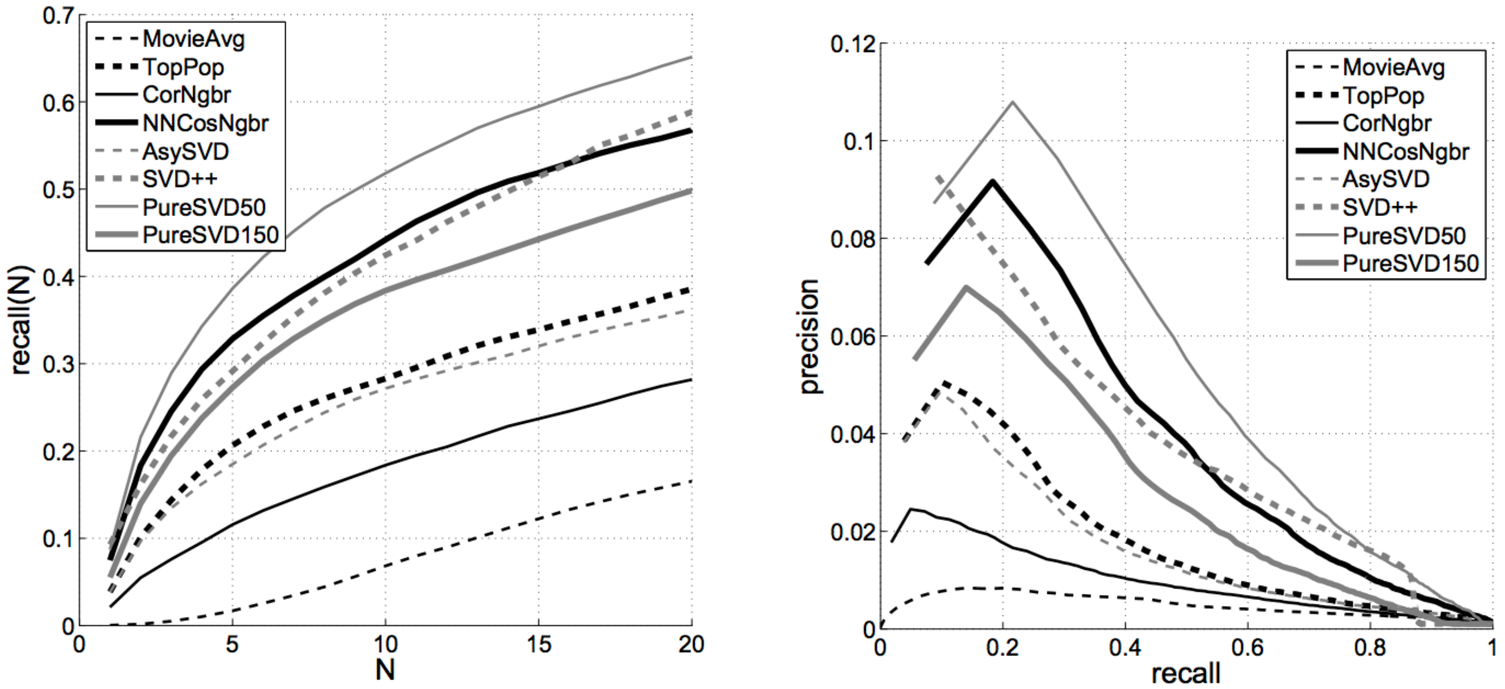


Rank **Evaluation**

- We can use the strategy followed by P. Cremonesi et.al.
- In order to measure the precision recall, first the models is trained using the training data, and then, for each item ***i*** rated with 5 stars in the test data set:
  - A set of 100 random unseen movies for the user of the item ***i*** are selected. We assume that these random movies will not be at the same interest than the 5 star movie
  - We predict the rating of the movie of item ***i*** and 100 random unseen movies.
  - We form a rank list by ordering all the 101 item according to the predicted rating. Let denote ***p*** the rank of the test item ***i*** within the list. The best results corresponds to the case the test item ***i*** precedes all the random items (i.e., p=1).
  - A top-N recommendation list by piking the **N** top ranked items from the list. If ***p≤N*** we have a hit. Otherwise we have a miss. Chances of hit increases as **N** is higher.

Performance of recommender algorithms on top-n recommendation tasks
P Cremonesi, Y Koren, R Turrin
Proceedings of the fourth ACM conference on Recommender systems, 39-46 612 2010

Rank **Evaluation**



(a) recall

(b) precision vs recall

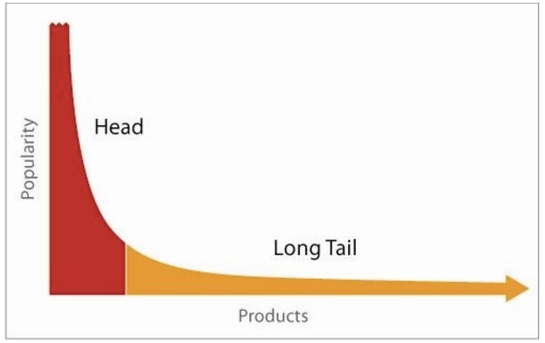
Figure 2: Movielens: (a) recall-at-N and (b) precision-versus-recall on all items.

Performance of recommender algorithms on top-n recommendation tasks
P Cremonesi, Y Koren, R Turrin
Proceedings of the fourth ACM conference on Recommender systems, 39-46
612 2010

Rank **Evaluation**

- The accuracy measures are usually heavily influence by rating from populars item. Item that received very few items are practically ignored.

- However, the non-popular items are typically the ones which provides higher profit to the companies.



***Trick**: weight items according to the profit, or remove the those items which are too popular.*

How to rank?

The problem:

It is **hard to optimize** a machine learning model using these the previous **measures** since they are **not differentiable**.

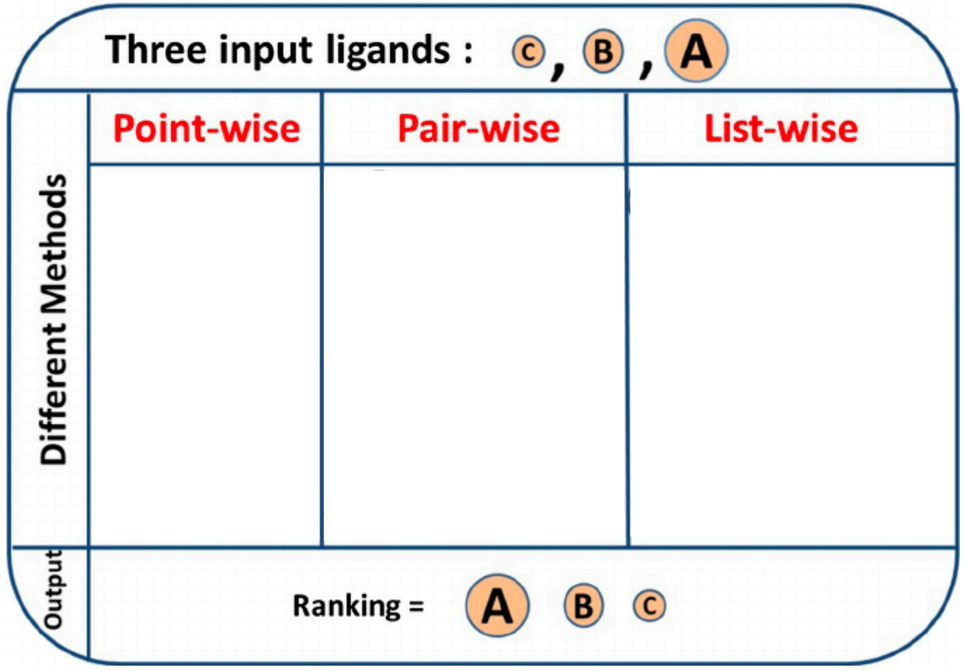
Learning to rank

- Optimization techniques that **learn rank-order directly instead of minimizing prediction error** are referred to as Learning-to-Rank (LTR).
- Three main approaches:
  - **Pointwise**:
    - Predict the absolute relevance (e.g. RMSE)
  - **Pairwise**:
    - Predict the ranking of a document pair (e.g. AUC)
  - **Listwise**:
    - Predict the ranking of a document list (e.g. cross entropy)

Learning to rank

- Using Machine Learning, the goal is to **construct a ranking model** from the training data.

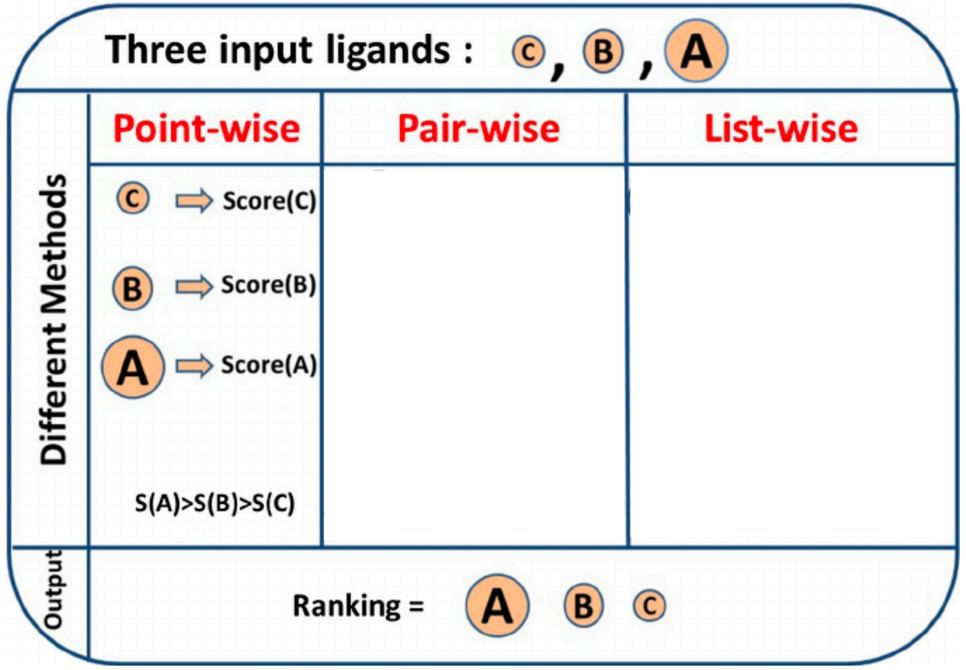
- The problem can be treat as a standard classification problem



Learning to rank

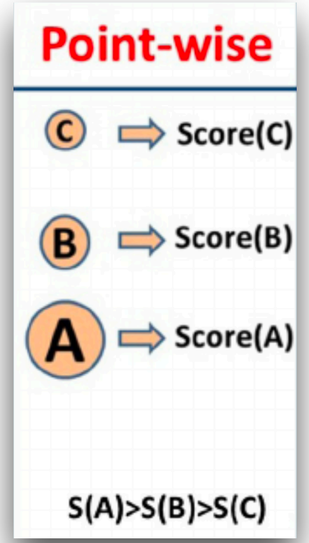
- Using Machine Learning, the goal is to **construct a ranking model** from the training data.

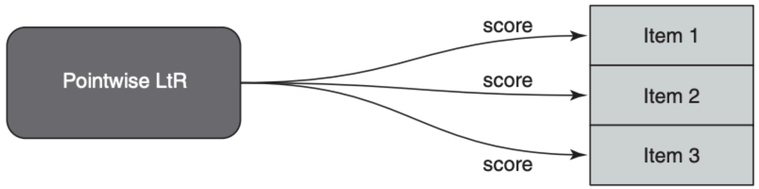
- The problem can be treat as a standard classification problem



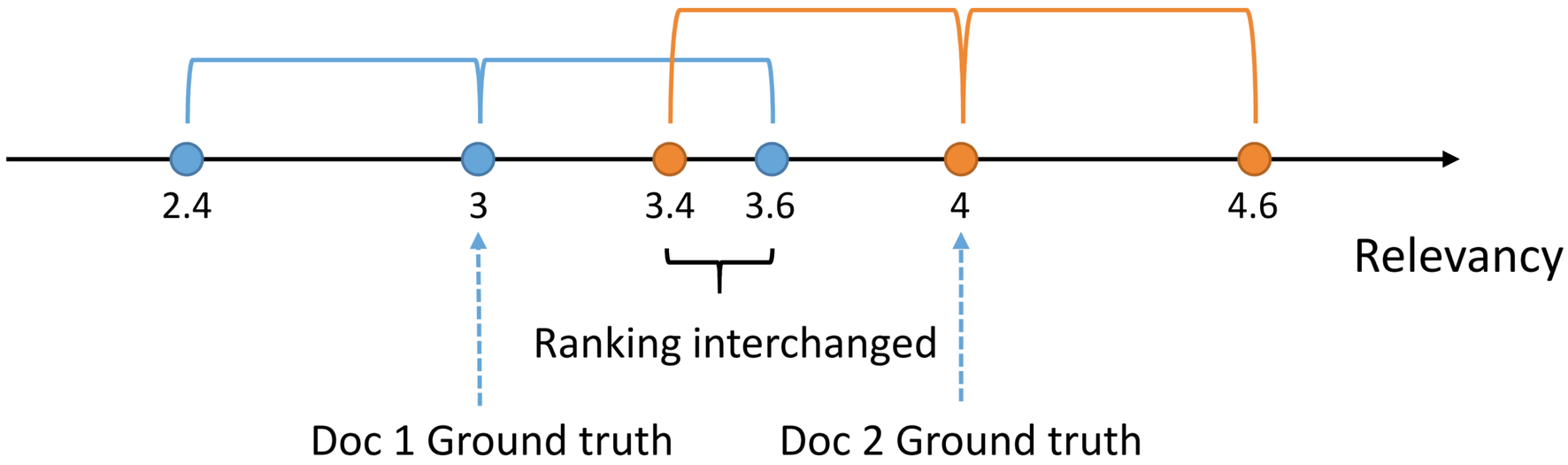
Learning to rank

- Pointwise approaches look at a **single item** at a time in the loss function. They essentially take a single item and train a classifier / regressor on it to predict how relevant it is for the current query/ user. The final ranking is achieved by simply sorting the result list by these document scores.
- For pointwise approaches, the score for each item is independent of the other items that are in the ranking list.
- All the standard regression and classification algorithms (SVM, XGBOOST, NN,…) can be directly used for pointwise learning to rank.
  - These methods tries are trained trying to minimize the error on the rating prediction… Usually RMSE, CE, or other similar loss function…
  - **Problem**: Obtained accuracy is highly biased to popular items.





Point Accuracy != Ranking Accuracy



Same square error might lead to different rankings

Learning to rank

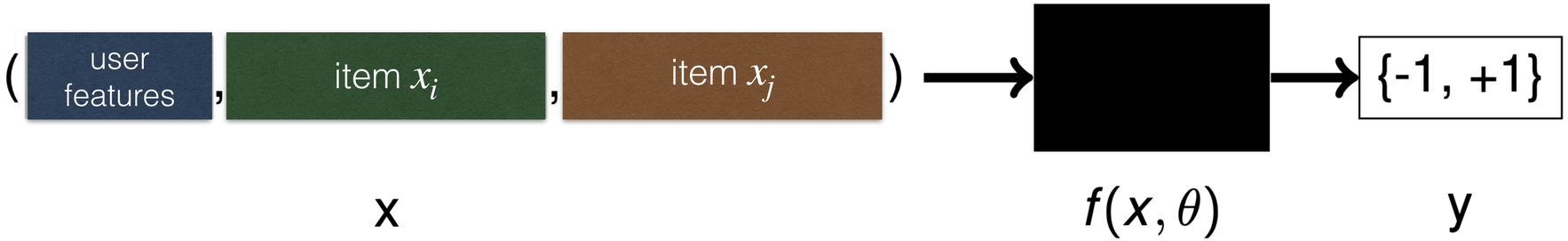
- **Pointwise** vs. **Pairwise/Listwise**
  - Train on pair or lists of training samples instead of individuals observations.
  - The loss functions are based on the relative ordering of items instead of the raw orders.
  - doesn’t care much about the exact score that each item gets, but cares more about the relative ordering among all the items.

Pairwise approach

Instead of focusing on recommendations as a rating prediction problem, it sometimes makes more sense to look at **how the items should be stacked -> relative preference**

Pairwise approach

- **General Criteria**: The ranking function f learns tor rank pairs of items (i.e. for $\{x_i, x_j\}$, is y_i greater than y_j?).



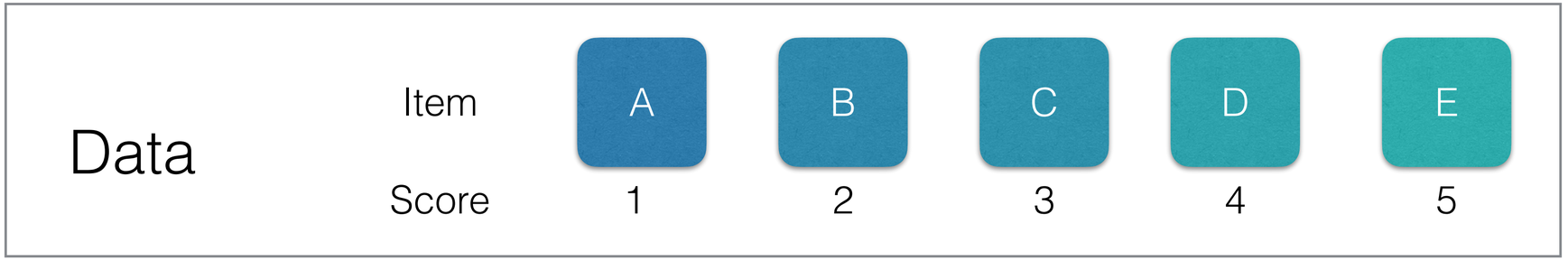
Pairwise approach

- **General Criteria**: The ranking function f learns to rank pairs of items (i.e. for $\{x_i, x_j\}$, is y_i greater than y_j?).
  - predict for every **pair of items** based on feature vector x
  - users' **relative preference** regarding the documents.
  - training determines the **parameter** θ based on a **loss function** (e.g. the number of inverted pairs)

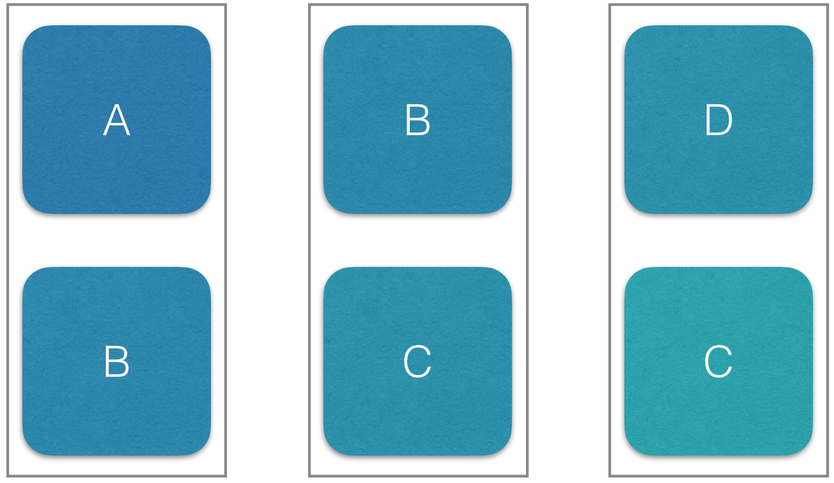
Advantage:
Models relative order

Main disadvantage:
- no distinction between excellent-bad and fair-bad
- sensitive tot noise labels.

Pairwise approach



- Training data instances are item pairs in learning



With labels:
+1 (i>j) or **-1** (j<i)

Pairwise approach

- **Several Methods**:
  - RANK SVM
  - RankBoost
  - RANK NET
  - Lambda Rank
  - Lambda Mart

Pairwise approach
RankNet

- RankNet was originally developed using neural nets.
- **The cost function for RankNet aims to minimize the number of *inversions* in ranking**. Here an inversion means an incorrect order among a pair of results, i.e. when we rank a lower rated result above a higher rated result in a ranked list. RankNet optimizes the cost function using Stochastic Gradient Descent.

Learning to Rank using Gradient Descent

Keywords: ranking, gradient descent, neural networks, probabilistic cost functions, internet search

Chris Burges CBURGES@MICROSOFT.COM
Tal Shaked* TAL.SHAKED@GMAIL.COM
Erin Renshaw ERINREN@MICROSOFT.COM
Microsoft Research, One Microsoft Way, Redmond, WA 98052-6399

Ari Lazier ARIEL@MICROSOFT.COM
Matt Deeds MADEEDS@MICROSOFT.COM
Nicole Hamilton NICHAM@MICROSOFT.COM
Greg Hullender GREGHULL@MICROSOFT.COM
Microsoft, One Microsoft Way, Redmond, WA 98052-6399

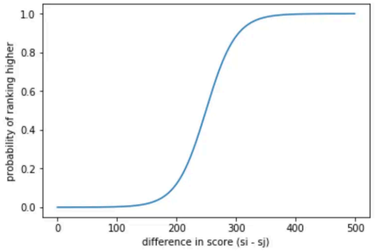
Pairwise approach
RankNet

- RankNet [Burges et al., 2005] is a pairwise loss function—popular choice for training neural L2R models and also an industry favorite [Burges, 2015]

- Predicted probabilities:

$$p_{i,j} = p(s_i > s_j) = \frac{1}{1 + e^{-\gamma(s_i - s_j)}}$$

$$p_{j,i} = \frac{1}{1 + e^{-\gamma(s_j - s_i)}}$$



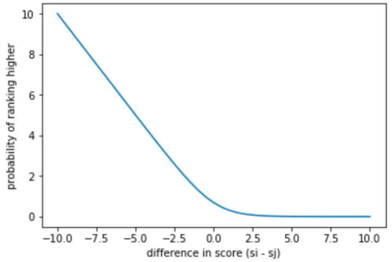
- Desired probabilities:

$$\bar{p}_{ij} = 1 \text{ and } \bar{p}_{ji} = 0$$

- Computing cross-entropy between $\bar{p}$ and p

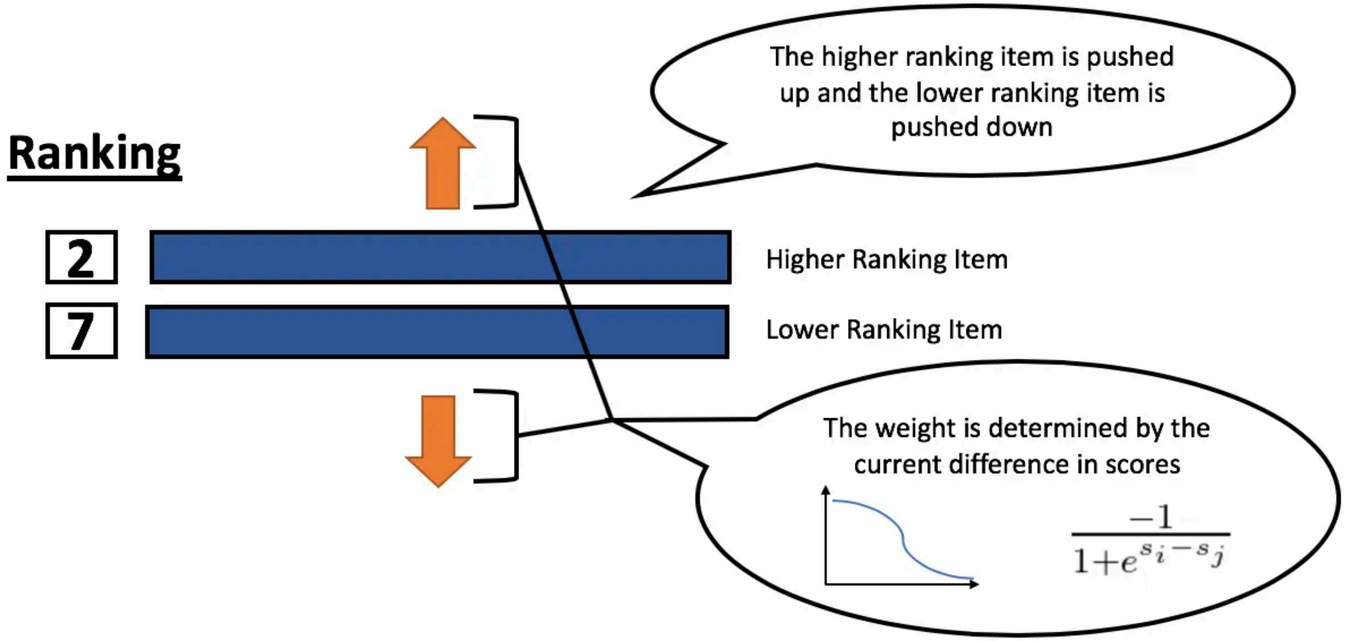
$$L_{RankNet} = -\bar{p}_{ij} log(p_{ij}) - \bar{p}_{ji} log(p_{ji}) = -log(p_{ij})$$

$$= log(1 + e^{-\gamma(s_i - s_j)})$$



Pairwise approach

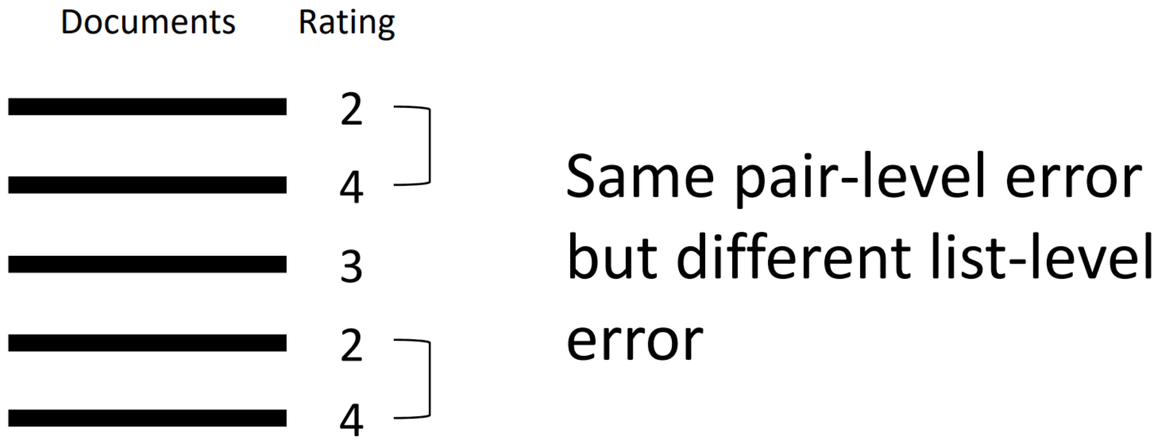
RankNet



Pairwise approach
RankNet

RankNet and is pretty good in many situations. There is one major pitfall to the RankNet approach though: the loss is agnostic to the **actual ranking** of the item.

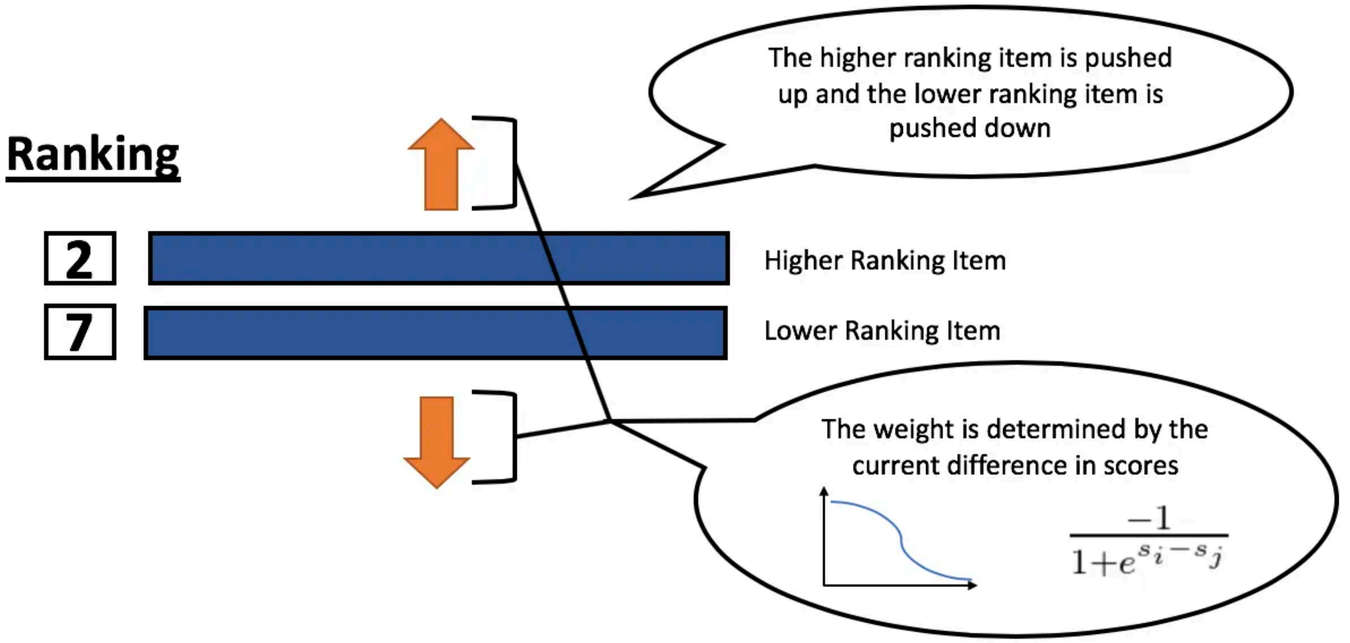
The strength of the push is determined only by the current difference in scores. So it doesn't matter if the items that rank below are 1, 2, or 500 ranks below



Pairwise approach
LambdaRank

- Burgess et. al. found that during RankNet training procedure, you don't need the costs, only need the gradients (λ) of the cost with respect to the model score. You can think of these gradients as little arrows attached to each document in the ranked list, indicating the direction we'd like those documents to move.

- Further they found that scaling the gradients by the change in NDCG found by swapping each pair of documents gave good results. The core idea of LambdaRank is to use this new cost function for training a RankNet. On experimental datasets, this shows both speed and accuracy improvements over the original RankNet.



Now, $\lambda_{ij} = \dfrac{-\Delta(i,j)}{1 + e^{(s_i - s_j)}}$ where $\Delta(i,j)$ is a penalty corresponding to how "bad" it is to rank items i and j in the wrong order.

Multiple metrics can be used to compute Δ, but the most common is the NDCG (normalized discounted cumulative gain)

Pairwise approach
Bayesian Personalized Ranking (BPR)

- With implicit feedback, the notion of "ratings" become less precise. As a result, more recent matrix factorization models start to move away from estimating a specific set of ratings to, instead, modeling the relative preferences (or orders) between different item

- Let Di be a set of item pairs (j, k) where user i has interacted with item j but not item k , assuming user i might be more interested in item j than item k, BPR minimizes the pairwise ranking loss

$$\min_{\mathbf{u}_*, \mathbf{v}_*} \sum_{i \in \mathcal{I}} \sum_{(j,k) \in \mathcal{D}_i} -log\ \sigma(\mathbf{u}_i^T \mathbf{v}_j - \mathbf{u}_i^T \mathbf{v}_k) + \lambda_u \|\mathbf{u}_i\|^2 + \lambda_v \|\mathbf{v}_j\|^2,$$

  - where σ is the sigmoid function.

- Problem: BPR loss does not sufficiently penalize the items that are at a lower rank

Pairwise approach
Bayesian Personalized Ranking (BPR)

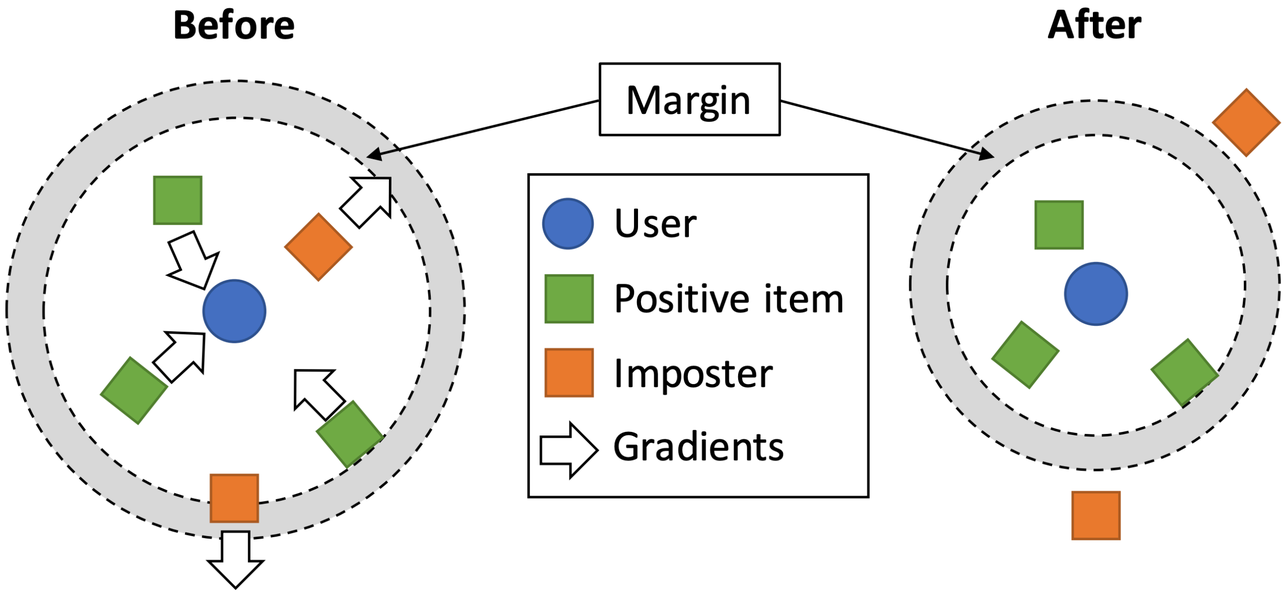
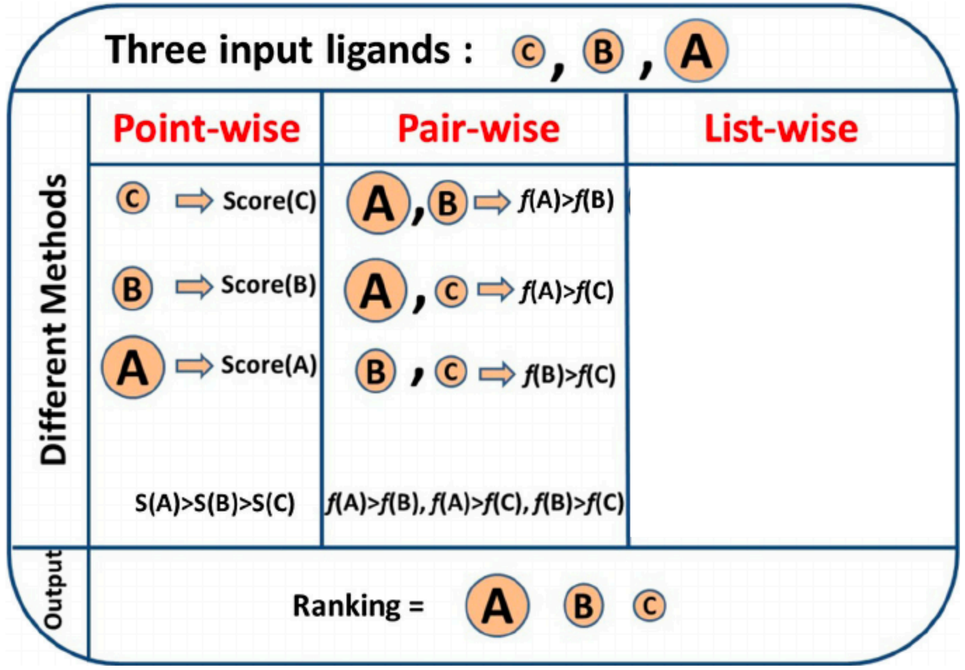


Figure 1: An illustration of collaborative metric learning. The hinge loss defined in Eq. 1 creates a gradient that *pulls* positive items closer to the user and *pushes* the intruding impostor items (i.e., items that the user did not like) away until they are beyond the safety margin.

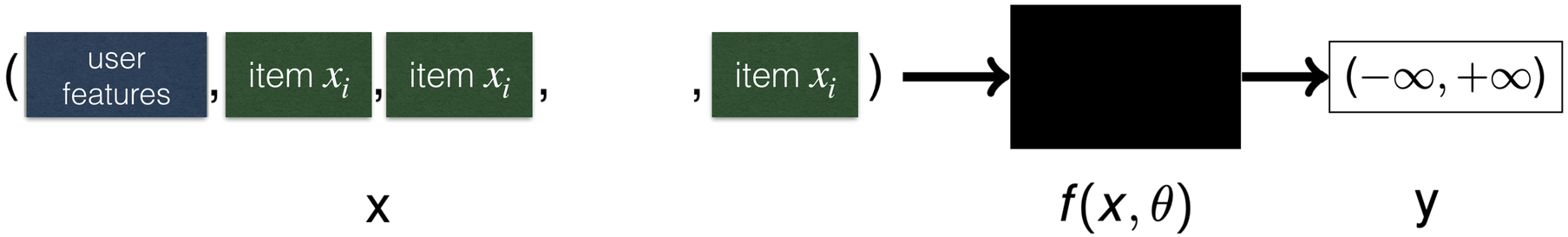
Learning to rank

- Using Machine Learning, the goal is to **construct a ranking model** from the training data.

- The problem can be treat as a standard classification problem



Listwise approach



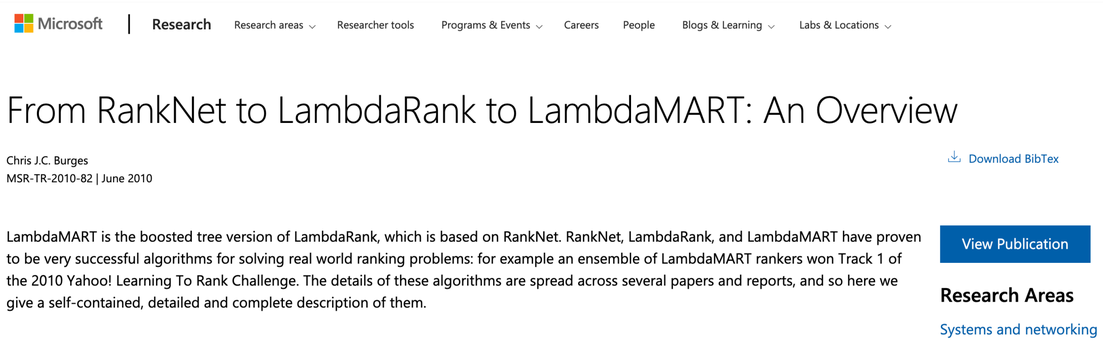
- predict for ranked list of documents based on feature vector x
- **effectiveness** of ranked list **y** (e.g., MAP or nDCG)
- training determines the **parameter** θ based on a **loss function**

Advantage:
positional infirmation visible to loss function

Main disadvantage:
- high training complexity

Lambda Mart

- LambdaMART combines LambdaRank and MART (Multiple Additive Regression Trees)
- MART uses gradient boosted decision trees for prediction tasks, LambdaMART uses gradient boosted decision trees using a cost function derived from LambdaRank for solving a ranking task

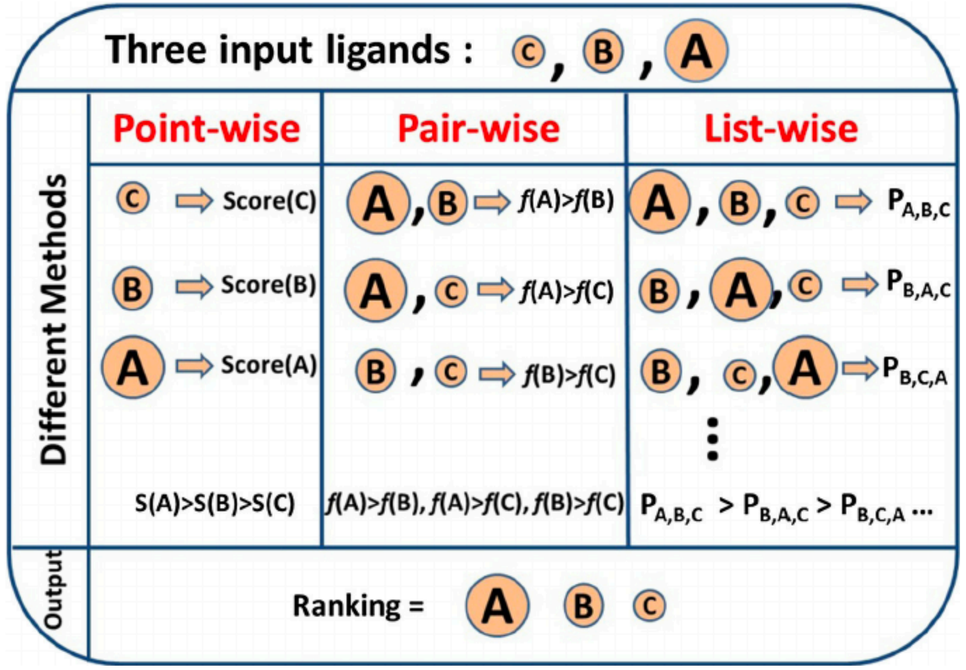


https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/

Learning to rank

- Using Machine Learning, the goal is to **construct a ranking model** from the training data.

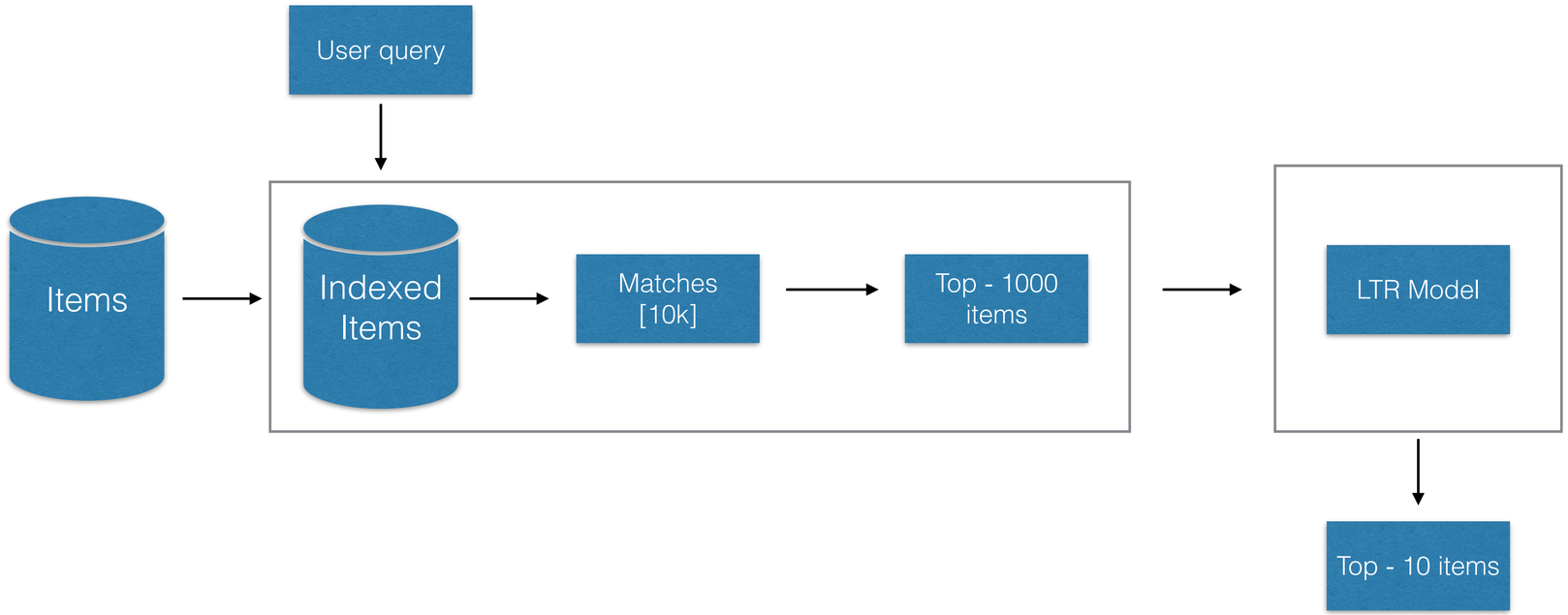
- The problem can be treat as a standard classification problem



LTR are great but
task consuming....

LTR framework

- LTR methods are computationally expensive.
- Usually used as re-rankers



If you have implicit data

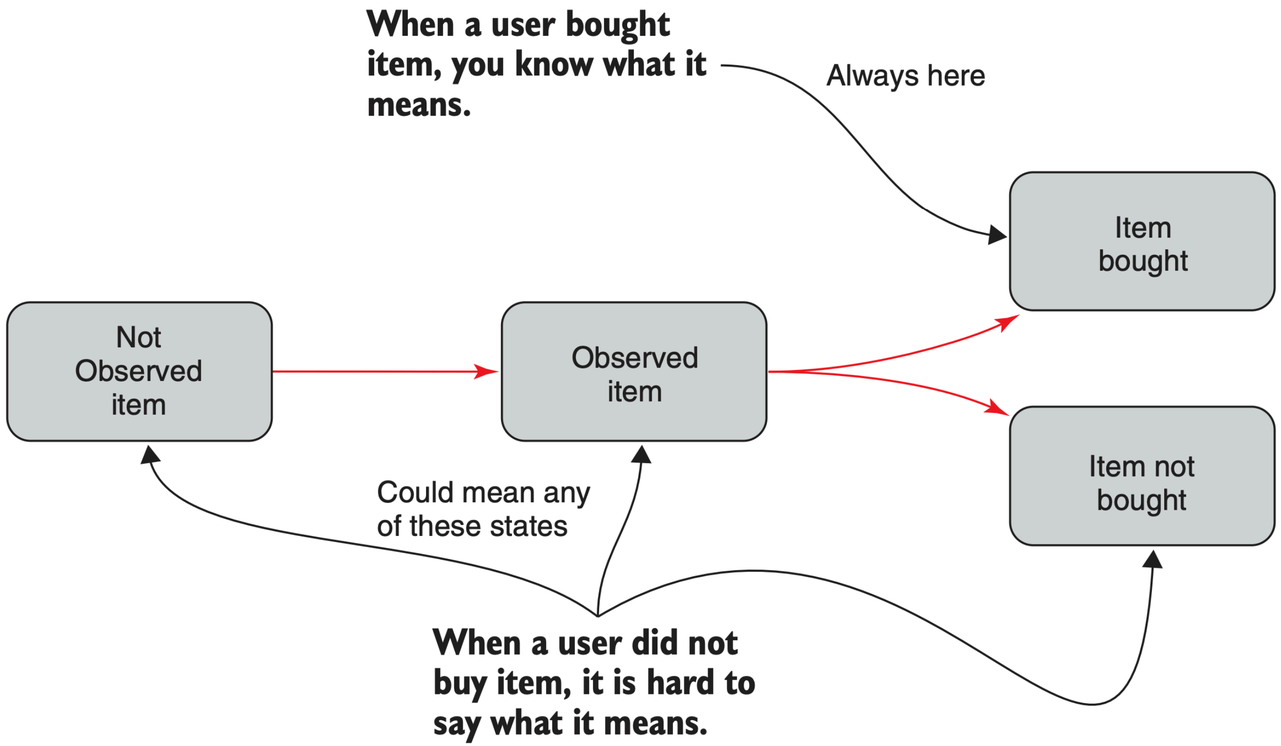


Figure 13.6 Different states of a user-item relationship. You know that when a user buys an item it's bought, but if a user doesn't buy an item, what does that mean?