Chapter IR:III

III. Retrieval Models

Learning to Rank

Motivation

Traditional IR Models

- Generative models
  - Learn the joint probability $P(q, d)$ of query and document(s)
  - i.e., TFIDF similarity, BM25 score, Naive Bayes probability, . . .
- Use a very small number of features
  - Term frequency
  - Inverse document frequency
  - Document length
- Few features, few parameters; can be tuned manually

But what if we want to exploit many features?

Learning to Rank Models

- Discriminative models
  - Learn the conditional probability $P(d|q)$ of document(s) given a query
  - Distinguish the decision boundary `relevant` vs. `non-relevant`
  - i.e., classification confidence, regression score, . . .
- Use a large number of features
  - Document-query features (term overlap, query term importance, ...)
  - Document-only features (images, links, length, recency, ...)
  - User feedback (click data, dwell times, eye tracking, ...)
- Many features, many parameters; have to be learned from data

ML for IR!

Learning to Rank
Formalization

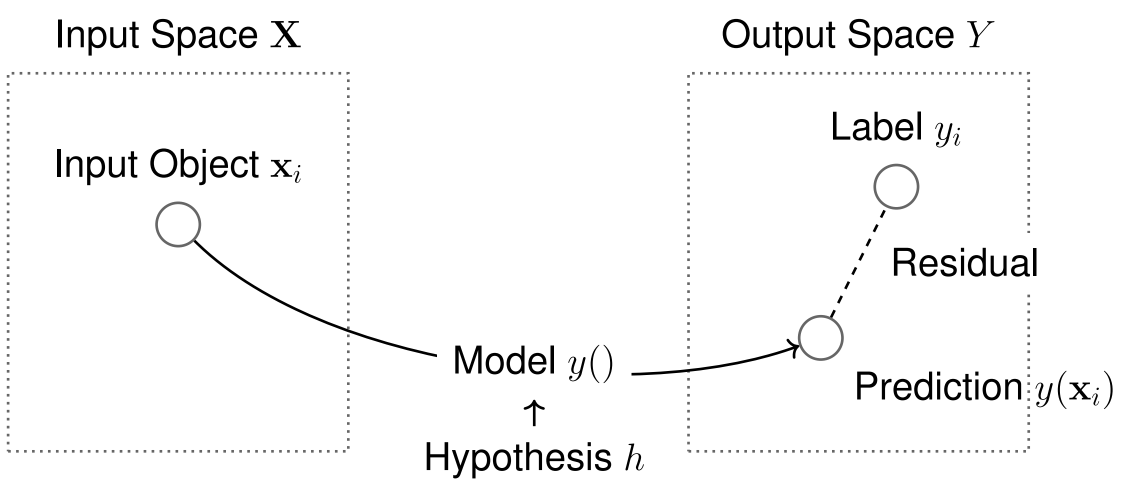
Learning to Rank (LTR) refers to supervised, feature-based, discriminative learning methods for IR. [Liu 2011]

- Supervised: based on training data with ground-truth relevance labels
- Feature-based: documents are represented by feature vectors
- Discriminative: relevance is estimated directly from observed features

Learning to Rank
Formalization

LTR tackles the ranking problem using machine learning techniques. This includes:

- Input space $\mathbf{X} \subseteq \mathbb{R}^n$, i.e., feature vectors $\mathbf{x}_i$ of query document pairs
- Output space $Y \subseteq \mathbb{R}$, i.e., relevance scores y_i of documents
- Model y, i.e., function mapping from input to outspput space
- Hypothesis space, i.e., parametrizations h of the model function
- Loss function ℓ, i.e., a measure to interpret the residual between the prediction $y(\mathbf{x}_i)$ and label y_i to choose an optimal h

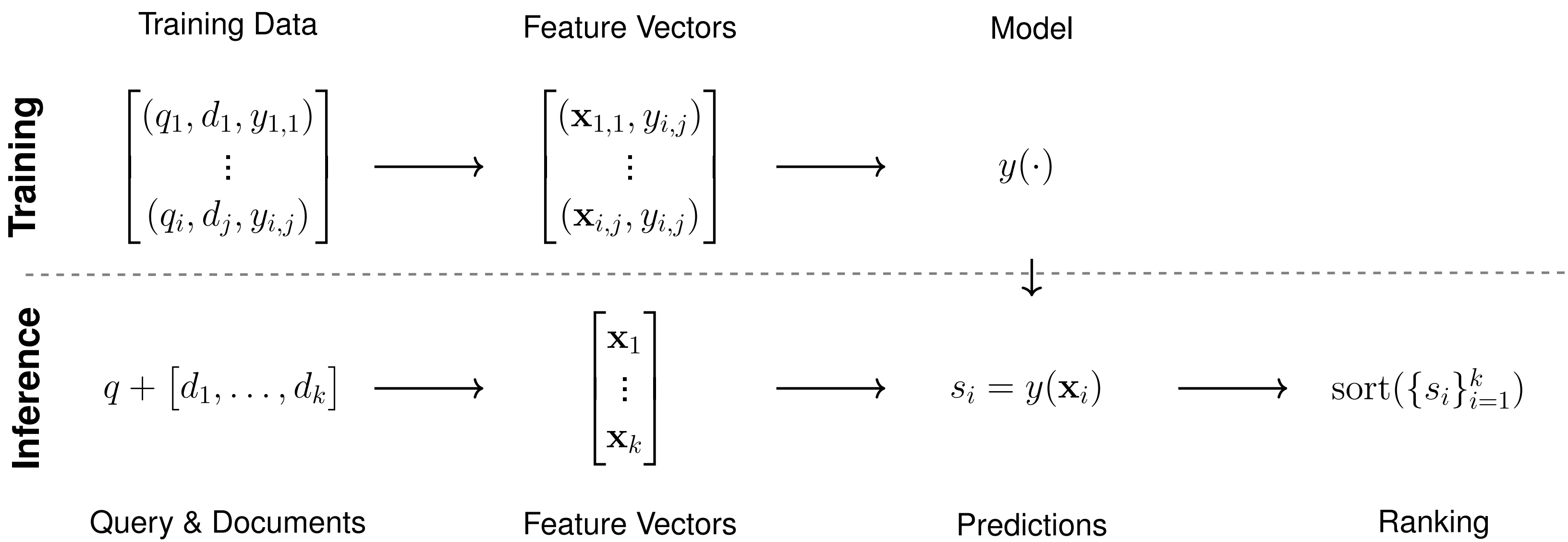


Learning to Rank

Components

Training an LTR system requires three components:

- a ground truth source to obtain training data from (supervised)
- a feature extraction from query-document pairs (feature-based)
- a model architecture to train for relevance prediction (discriminative)



Where do we get training data from?

Learning to Rank

Relevance Feedback

How can supervised ground truth data for relevance be collected?

- Explicit relevance feedback
  - Asking a user whether a result is relevant/non-relevant to a query
  - Obtrusive to users, expensive if tasked, hard to scale
  - Requires a group of assessors!

- Implicit relevance feedback
  - Predicting relevance based on user interactions (clicks)
  - Non-obtrusive, inexpensive, lots of data
  - Requires a live system with users!

But are clicks a reliable form of ground truth?

Learning to Rank

Mining Training Data From Clicks [Joachims et al., 2005]

- How do users behave?
  - Users tend to look close to where they click – clicks are guided by presented content
  - They view higher-ranks before clicking on a result – rankings are evaluated sequentially

- What are clicks influenced by?
  - Relevance influence: reversed rankings have more clicks at low ranks
  - Position influence: users tend to click on higher ranks even when lower ranks are more relevant

- What does that mean for LTR training data?
  - Clicks should not be used to derive absolute relevance judgements
  - Clicks can be used to derive pairwise preference judgements – a clicked result is more relevant than all higher ranked results that were skipped

Learning to Rank

Feature Extraction

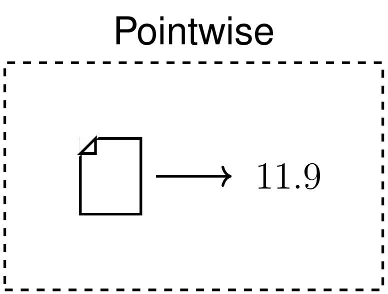
Different kinds of features:

- Query features
  - Content features (query intent classification, performance prediction, ...)
  - Metadata features (time of day/month/year, ...)
- Document features
  - Content features (spam/quality scoring, text classification models, ...)
  - Metadata features (number of slashes in URL, timestamps, ...)
  - Link features (PageRank, number of links, number of child pages, ...)
- Query-document features
  - Scores of other retrieval models (BM25, TFIDF, ...)
  - Term matching (edit distances, occurence scores, ...)
- User behaviour features
  - session data
  - user profiling

Learning to Rank
Training Tasks

Recap: what kinds of ground truth data are available for LTR?

- Pointwise: a single feature vecotr and its absolute relevance score

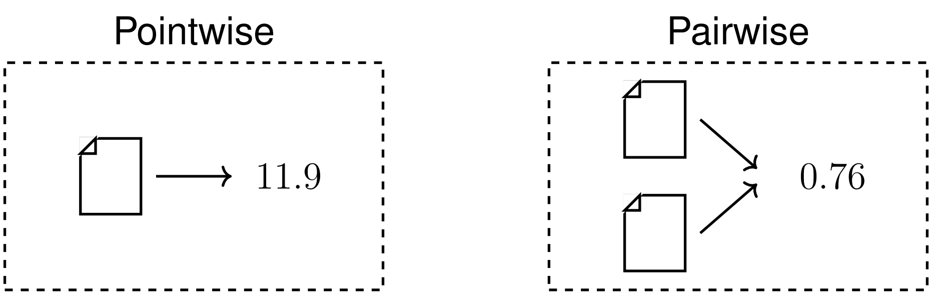


Learning to Rank

Training Tasks

Recap: what kinds of ground truth data are available for LTR?

- Pointwise: a single feature vecotr and its absolute relevance score
- Pairwise: a pair of feature vectors and a preference between them

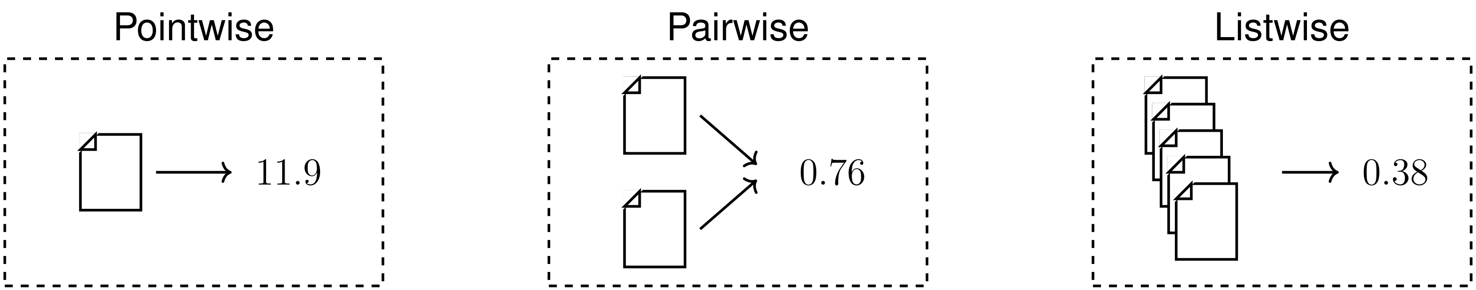


Learning to Rank

Training Tasks

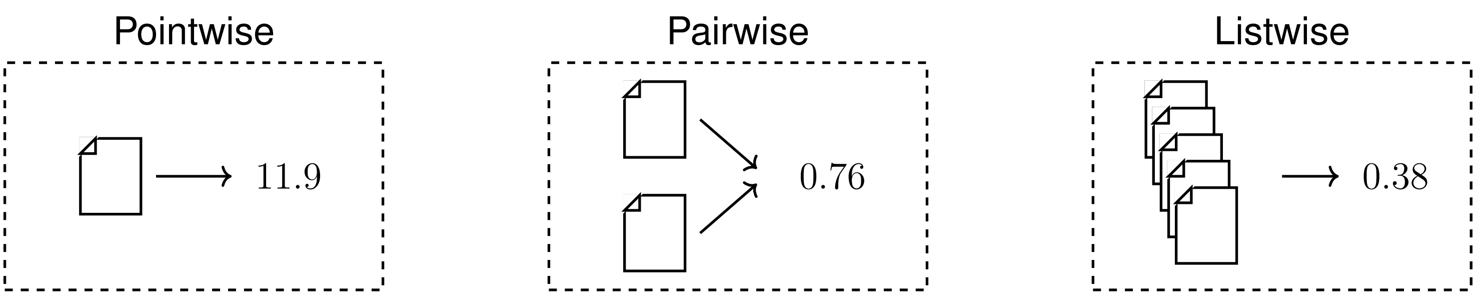
Recap: what kinds of ground truth data are available for LTR?

- Pointwise: a single feature vecotr and its absolute relevance score
- Pairwise: a pair of feature vectors and a preference between them
- Listwise: a ranking of feature vectors and its effectiveness



Recap: what kinds of ground truth data are available for LTR?

- Pointwise: a single feature vecotr and its absolute relevance score
- Pairwise: a pair of feature vectors and a preference between them
- Listwise: a ranking of feature vectors and its effectiveness



Each kind can be used to define a loss function for an LTR system!

Learning to Rank

Pointwise Loss

- A pointwise loss . . .
  - . . . operates on a single feature vector $\mathbf{x}_i$
  - . . . quantifies the error between predicted relevance and ground-truth relevance of document d_i
  - . . . takes (q, d_i, y_i) triples of query, document, and relevance as training instances

- Relevance estimations are absolute
  - scores are invariant w.r.t. strictly monotonous transformations (shifting/scaling all scores results in the same ranking)
  - absolute estimation is not as robust as relative estimation (small score changes might result in large rank changes)

- Relevance estimations are independent
  - only as single document is used to infer a relevance value
  - all other potential documents in the collection are ignored

Remarks:

- Pointwise LTR can alternatively be operationalized as classification (ranking by probability of belonging to class 'relevant'; $P(y(\mathbf{x}_i) = 1|q)$) or ordinal regression (membership of an ordered relevance class; $y(\mathbf{x}_i) \in [0, 1, \ldots, k]$) [Liu 2011]

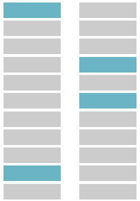
Learning to Rank

Pairwise Loss

- A pairwise loss . . .
  - . . . operates on a pair of feature vectors $(\mathbf{x}_i, \mathbf{x}_j)$
  - . . . quantifies the error of pairwise comparisons as indicated predicted relevances s_i and s_j and ground-truth comparison
  - . . . takes 4-tuples (q, d_i, d_j, y_i) of query, documents, and preference as training instances

- Relevance estimations are relative (take other documents into account)

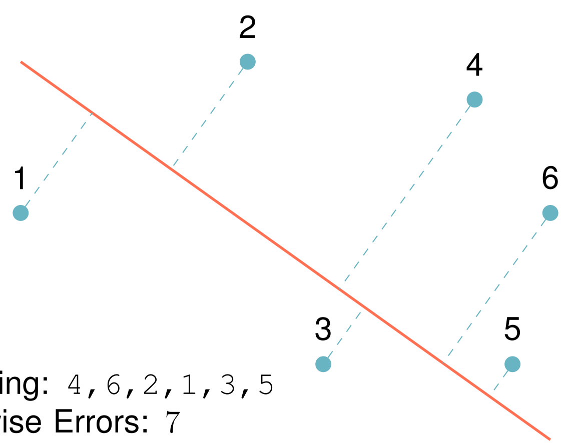
- Problem: comparison errors are not equally important at all ranks in practice
  - blue are relevant, gray are irrelevant
  - same number of pairwise errors (7)
  - e.g. nDCG is higher for left ranking



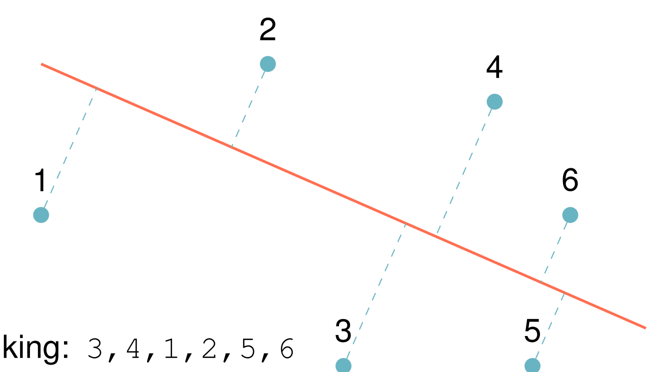
Learning to Rank

Pairwise Example – RankSVM [Joachims 2002]

- Idea: Learn a ranking function so that the number of violated pairwise training preferences is minimized
- Ranking function: margin distance to hyperplane $\mathbf{w}$; select $\mathbf{w}$ such that $y(\mathbf{x}_i) > y(\mathbf{x}_j) \iff d_i \succ d_j$ where the order is given by the training data
- Example: 2 features, points are training documents with their ground-truth rank; two different parametrizations for $\mathbf{w}$ shown.



Ranking: `4,6,2,1,3,5`
Pairwise Errors: 7



Ranking: `3,4,1,2,5,6`
Pairwise Errors: 4

Learning to Rank

Pairwise Inference (?)

Why not infer pairwise scores?

- Obtained pairwise comparisons are independent
  - Outcome of $y(\mathbf{x}_i, \mathbf{x}_j)$ is not dependent of, e.g., $y(\mathbf{x}_j, \mathbf{x}_i)$ or $y(\mathbf{x}_i, \mathbf{x}_k)$
  - Possibly inconsistent w.r.t. complementarity ($y(\mathbf{x}_i, \mathbf{x}_j) \neq 1 - y(\mathbf{x}_j, \mathbf{x}_i)$) or transitivity ($y(\mathbf{x}_i, \mathbf{x}_j) > 0.5 \wedge y(\mathbf{x}_j, \mathbf{x}_k) > 0.5 \wedge y(\mathbf{x}_k, \mathbf{x}_i) > 0.5$)
- Post-processing needed to convert comparison scores into a ranking
  - Sorting methods require total order, incompatible with inconsistencies
  - Ranking can be statistically approximated from inconsistent pairs
- Computational complexity is quadratic w.r.t. document count
  - For k documents, at worst $k(k-1)$ comparisons have to be made
  - Sampled comparisons for reduced complexity increase uncertainty

Learning to Rank

Listwise Loss

- A listwise loss . . .

  . . . operates on a sequence of feature vectors $[\mathbf{x}_1, \ldots, \mathbf{x}_i]$

  . . . quantifies the error of the ranking given by $s_1, \ldots, s_i$ with a ranking metric

  . . . takes $(q, (d_1, y_1), ..., (d_k, y_k))$ as training samples, i.e., a query and a sequence of document-relevance pairs, for which the metric is calculated

- Problem: Ranking metrics are non-differentiable w.r.t. model parameters
  - sort operator is non-smooth
  - change in parameters might not produce a different ranking, thus optimization is not possible on the ranking metric's score
  - instead, proxy metrics can be used that resemble the original metric with modifications to establish differentiability

Learning to Rank

Listwise Example – LambdaMART [Wu et al. 2010]

- Intuition: we do not need to explicitly define a smooth cost function, we only need to know its gradients (how does it change w.r.t to its input)
  - if a ranking metric changes a lot if we modify the rank of a document, the document should be ranked high
  - higher document relevance leads to higher impact w.r.t. ranking changes
  - We can use λ directly to rank documents!

- to optimize a metric M, we can just calculate the change λ_i of M for each $\mathbf{x}_i$ while modifying the ranking, i.e. swapping $\mathbf{x}_i$ with another feature vector

- optimization target becomes cumulative score change for all swaps
  - positive gradient – document is pushed up the ranking; negative gradient – document is pushed down the ranking
  - any metric M can be used as target, even non-smooth ones; ususally, nDCG is optimized

Learning to Rank

Listwise Example – LambdaMART

Given a query q and a set of documents with their relevance labels $\{(d_i, y_i)\}_{i=1}^k$:

1. Compute $\Delta_{ij}M$, the change in metric M if documents d_i and d_j with scores s_i and s_j are swapped; value is rescaled by predicted score difference; sign of value depends on ordering implied by ground-truth labels

$$\lambda_{ij} = S_{ij} \left| \Delta_{ij} M \frac{-1}{1 + e^{s_i - s_j}} \right|, \; S_{ij} = \begin{cases} 1 & y_i \geq y_j \\ -1 & y_i < y_j \end{cases}$$

2. gradient λ_i of a document is the sum of its metric value changes

$$\lambda_i = \sum_{j=0, i \neq j}^{k} \lambda_{ij}$$

3. Train a gradient boosted tree model (MART) to predict λ_i given features $\mathbf{x}_i$

Learning to Rank

Listwise Inference (?)

Why not infer the ranking directly?

- Predict a score for a given ranking?
  - Given a permutation of documents, predict its effectiveness
  - Every possible permutation would have to be scored to find optimal one
  - Input space is combinatorial ($k!$ for k documents) $\rightarrow$ Not feasible!
- Directly predict the ranking?
  - Given a set of documents, predict the indices of their optimal ordering
  - Model needs to be invariant to rearranging the input $\forall \sigma \in S_k : y(\sigma((\mathbf{x}_1, ..., \mathbf{x}_k))) = y((\mathbf{x}_1, ..., \mathbf{x}_k))$
  - Output space is combinatorial ($k!$ for k documents) $\rightarrow$ Not feasible!

Learning to Rank

Comparison of Approaches

	Pointwise	Pairwise	Listwise
Loss	Single Doc.	Doc. Pair	Doc. Ranking
Relevance	Absolute	Relative	Relative
Effectiveness	Good	Better	Best
Complexity	Low	Medium	High

Rank Fusion

Overview

Combine different systems or ranking functions into a single ranked list. [Wu 2012]

- Collection of documents D
- All retrieval systems execute a query q on D
- Final set of rankings $r_i \in R$ from each retrieval system $r_i = \langle d_{i_1}, d_{i_2}, ..., d_{i_m} \rangle$
- Fusion method produces a final ranking from set of rankings R

Rank Fusion

Overview

Combine different systems or ranking functions into a single ranked list. [Wu 2012]

- Collection of documents D
- All retrieval systems execute a query q on D
- Final set of rankings $r_i \in R$ from each retrieval system $r_i = \langle d_{i_1}, d_{i_2}, ..., d_{i_m} \rangle$
- Fusion method produces a final ranking from set of rankings R

Two methods of fusion:

- Score-based
- Rank-based

Rank Fusion

Score-based

Score-based rank fusion's aim is to provide a global score for a document $g(R, d)$

CombSUM

- Global score computed by summing relevance score ρ of document across R

$$g(R, d) = \sum_{r_i \in R} s(d, r_i)$$

- If $d \notin r_i$, then $s(d, r_i) = 0$

Rank Fusion
Score-based

Score-based rank fusion's aim is to provide a global score for a document $g(R, d)$

CombSUM

- Global score computed by summing relevance score ρ of document across R

$$g(R, d) = \sum_{r_i \in R} s(d, r_i)$$

- If $d \notin r_i$, then $s(d, r_i) = 0$

CombMNZ

- Summed score multiplied by times document appears across all rankings

$$g(R, d) = |d \in R| \sum_{r_i \in R} s(d, r_i)$$

- No default score if $d \notin R$

Rank Fusion

Score-based

Score-based rank fusion's aim is to provide a global score for a document $g(R, d)$

CombSUM

- Global score computed by summing relevance score ρ of document across R

$$g(R, d) = \sum_{r_i \in R} s(d, r_i)$$

- If $d \notin r_i$, then $s(d, r_i) = 0$

CombMNZ

- Summed score multiplied by times document appears across all rankings

$$g(R, d) = |d \in R| \sum_{r_i \in R} s(d, r_i)$$

- No default score if $d \notin R$

It is important to normalise the scores for each r_i. Why?

Rank Fusion

Score-based with Learning To Rank

Learn the weights for each ranker under CombSUM:

- Each ranker is assigned a weight w_i, linearly combine the weights

$$g(R, d) = \sum_{r_i \in R} w_i \cdot s(d, r_i)$$

- Heuristics weights, e.g., query performance prediction (QPP)

- Performance weights, e.g., grid search using nDCG$@k$ over representative training queries

- Learned weights, e.g., consider scores from rankers as features and apply LTR

Rank Fusion

Rank-based

Re-rank documents according to their rank position in r_i, ignoring relevance scores.

Borda Count

- Score of a document is the number of documents ranked lower (Election voting algorithm)

$$g(R, d) = \sum_{r_i \in R} \frac{|r_i| - \text{rank}(r_i, d) + 1}{|r_i|}$$

- Conceptually the same as CombSUM, but uses document positions instead of scores

- Can be used if scores are low quality or not available

Rank Fusion

Intuitions

Intuition behind success of rank fusion: [Vogt 1999]

- Chorus Effect ➜ Multiple retrieval approaches suggest that a document is relevant to a query

- Dark Horse Effect ➜ One retrieval approach suggests document is very relevant while not retrieved by other approaches

Which of these effects are lessened or boosted by CombMNZ?
What about other rank fusion methods?