Summary Information Retrieval 1

Phillip Lippe

March 23, 2021

Contents

1 The indexing architecture

1.1 Statistical Properties of Text

- Zipf's law: frequency of word is inversely proportional to its rank (deviates for high frequency and low frequency words)
- Heaps' law: number of unique words is sublinear in number of words in collection

1.2 Text analysis pipeline

1. remove whitespace and punctuation
2. convert terms to lower-case
3. remove stop-words (words that occur in every document -¿ are not characteristic for document)
4. convert terms to stems (e.g. drop suffixes etc.)
5. deal with phrases

- how to determine that word is stop word?: either through frequency threshold or by using dictionary (or using a combination of both)
- different approaches to stemming:
  - porter stemmer is a list of manually created rules
  - dictionary-based stemming: has new-words problems
  - hybrid approach (e.g. krovetz stemmer). uses dictionary-based approach first. if word is not found -¿ apply algorithmic stemmer -¿ check dictionary again -¿ if not found again apply rule and check dictionary again ...

1.3 Web graph

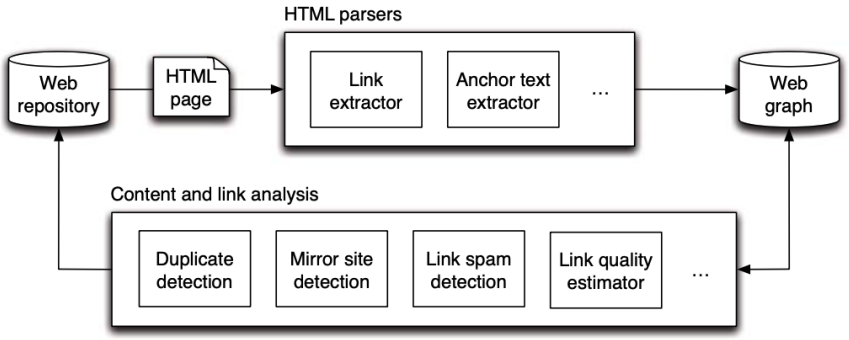


Figure 1: Web graph

- see collection as graph where documents are nodes and links are edges

1.4 Forward index

- anchor text comes from links
- forward index: for every document store all informtion about the document (e.g. all the words that occur etc.)

1.5 Page attribute file

- Contains information such as language, length, text quality, link quality etc.
- whatever data describes the document, put it into a separate attribute file

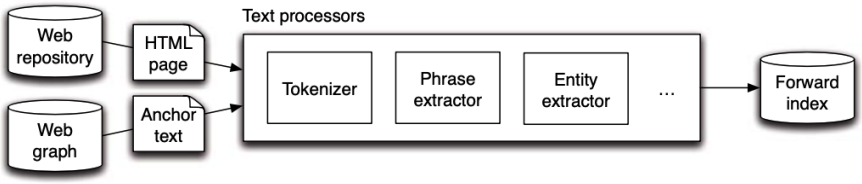


Figure 2: Forward index

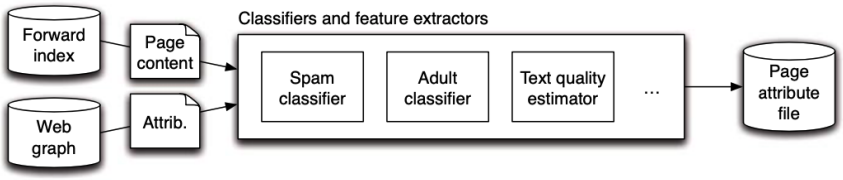


Figure 3: Page attribute file

1.6 Inverted index

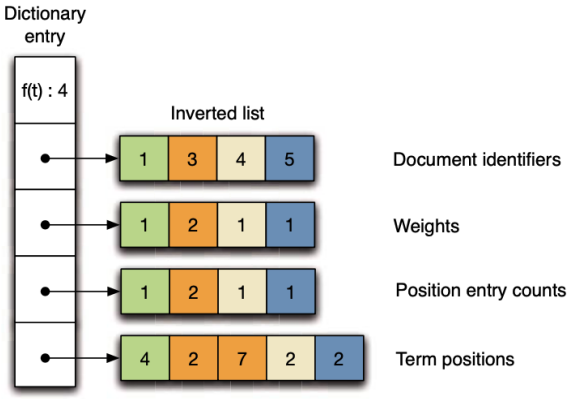


Figure 4: Full inverted index

- needed for search: from words to documents (e.g. index at the end of textbook)
- Inverted index is a dictionary where each entry contains total number of docs containing the term, pointer to start of the inverted list (which contains the document identifiers), other meta-data about the term
- the actual inverted list could be modelled by a B+-tree or hash table
- inverted list can also contain term-frequency per document or positions of the terms
- in reality, store all kinds of information in different inverted lists (e.g. one list for document identifiers, one list for term positions etc.)

See Fig. 4.

1.7 Constructing inverted indices

A simple construction algorithm that iterates through the documents and constructs an inverted index along the way has two problems: it is in-memory and is single-threaded. Possible alternatives are:

1.7.1 Two-pass index

Addresses the in-memory issue.

- goes throught he collection two times

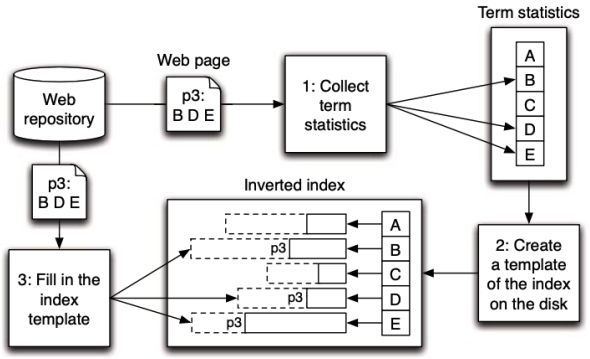


Figure 5: Two-pass index

- first time, it computes statistics: for term A there are 10 documents etc.
- based on these statistics, empty index is created on disk
- during second pass, fill in the created indices
- -¿ use no RAM

1.7.2 One-pass index with merging

Addresses the in-memory issue.

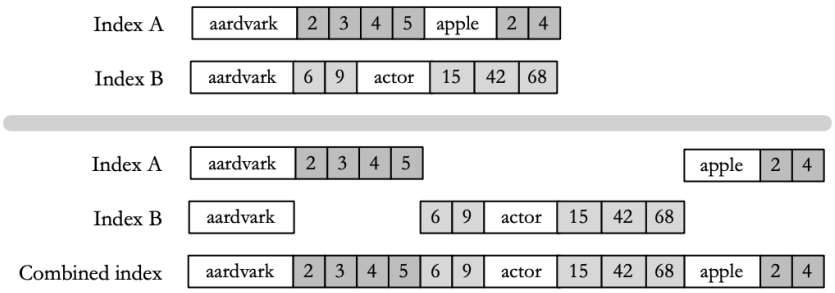


Figure 6: One-pass index with merging

- construct multiple indices over possibly different documents until memory is full
- then merge the indices
- in some sense this is two passes as well

1.7.3 Distributed indexing

Addresses the single-threaded issue.

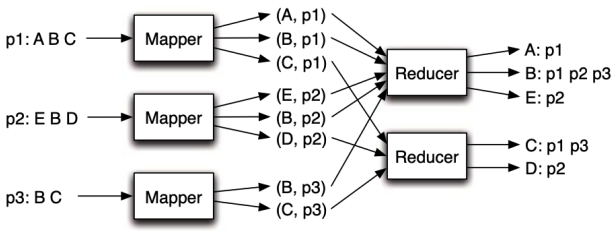


Figure 7: Distributed indexing (MapReduce)

- two types of processes: maps and reduces

- take one mapper for each document -¿ process documents in parallel
- mapper produces pair of key = term and value = document
- reducers collect entries that have the same key and produces list of documents for each key

1.8 Updating indices

1.8.1 No merge

- low index maintenance cost
- high query processing cost
- when encountering changes, add changes to delta index in memory
- when out of memory, flush delta to disc
- at query time, search all delta indices and merge

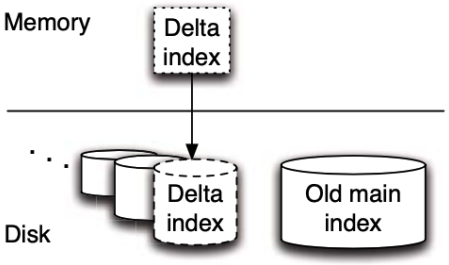


Figure 8: Updating indices with no merge

1.8.2 Incremental update

- keeps free buffer space
- no read/write of entire index when updating
- inverted lists are accessed concurrently as index querying and updating is done at the same time
- run out of free buffer space

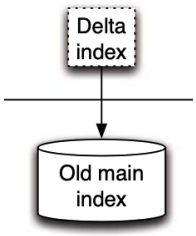


Figure 9: Updating indices with incremental updates

1.8.3 Immediate merge

- always a single index
- read/write of entire index when updating -¿ slow when having many documents
- query only main index

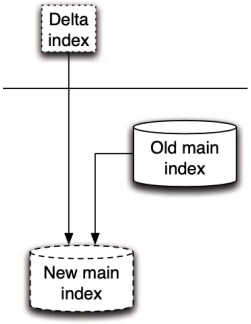


Figure 10: Updating indices with immediate merge

1.8.4 Lazy merge

- trade-off between index maintenance cost and query processing cost
- -¿ we have more than one index but not as many as first generation delta indices
- no concurrency
- -¿ practicals

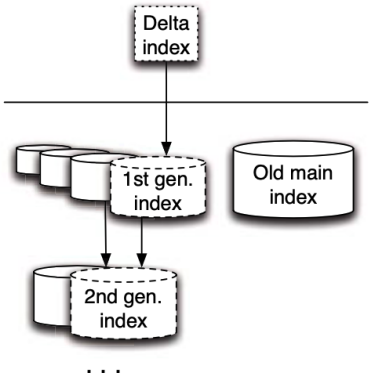


Figure 11: Updating indices with lazy merge

2 Offline evaluation

- Evaluating an IR system without any interaction with user
- Assumption: assessors can tell what is relevant

2.1 Collection-based evaluation

- Approximating user happiness by relevance of the found documents
- There are different measures to do that

2.1.1 Traditional Evaluation measures

- Relevance is assumed to be:
  - topical (document and query on the same topic)
  - binary (relevant / not relevant)
  - independent (relevance of document A does not depend on relevance of document B)
  - stable (relevance judgements do not change over time)

  - consistent (judgements are consistent across editors)
  - complete (no missing judgements)
- We can view IR as a (binary) classification problem where every document is either relevant or not with respect to a query
- The evaluation is performed by calculating precision ($\frac{TP}{TP+FP}$) and recall ($\frac{TP}{TP+FN}$)
- However, the output of an IR system is a ranking and not a binary classification. Thus, we label the first k documents the system proposes as relevant, and other as non-relevant $\implies$ precision/recall @ cut-off ($P@k/R@k$)
- Trade-off between precision and recall: returning more documents increases recall; returning fewer documents often increases precision
- The trade-off between precision and recall is task specific. For web searches, we want a high precision on the first few documents, but allow a worse recall (as a user doesn't want to find *all* relevant documents). In contrast, in medicine, we want to have a high recall to not miss an important document
- Another way to incorporate precision and recall for ranking is using R/P curves. We start with a cut-off point of 0 where precision is 1 and recall 0. Then we increase the cut-off point one by one and record the new values in the curve (left, Figure 12). While the cut-off rank heads to infinity, recall goes to 1 and precision to 0. We interpolate the curves by propagating maximum to the left.

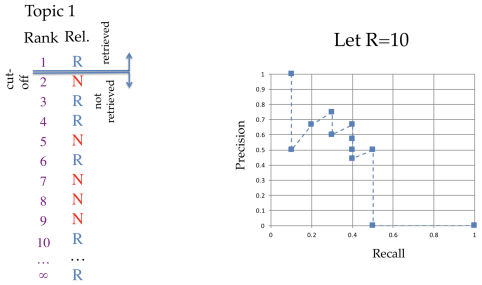


Figure 12: R/P curves for ranking

- When having multiple queries, we would average the RP curves.
- The area under the curve is the average precision which can also be calculated by taking the average of all precision values at ranks with relevant documents.
- Usually, a binary scale of whether a document is relevant is not sufficient. For a graded relevance scale, we can use different evaluation measures
  - **Discounted Cumulative Gain (DCG)** - considers the relevance grade and position of every document. The total gain is accumulated at a certain rank k:

$$DCG@k = \sum_{\text{rank}r=1}^{k} \frac{2^{\text{rel}_r} - 1}{\log_2(1+r)}$$

  - The numerator is the non-linear relevance score of the document at rank r, and the denominator the discount over ranking position (assumes less utility for the user when document is ranked lower as it is less likely to be examined)
  - The score highly depends on the best possible ranking for a query. Thus, the DCG can be normalized by the value of the best ranking $\implies 0 \leq nDCG \leq 1$. This makes it easier to compare scores over different queries
- When having only a single relevant document (e.g. navigation), then recall-based measures not appropriate. Instead, the search time for the user is determined by the ranking of the matching result. Therefore, optimize for high ranking of that result e.g. by using reciprocal rank (1/rank of (first) relevant document)

2.1.2 Model-based Evaluation measures

- Another perspective of evaluation is looking at different aspects of possibles metrics. A model-based approach considers the following three components:

  1. **Browsing model** - describes how the user interacts with results, like the probability of a document being clicked/viewed $\Rightarrow p(d)$
  2. **Model of document utility** - describes how a user derives utility from individual relevant documents. Similar to how to determine the graded relevance scale $\Rightarrow g(d)$.
  3. **Utility accumulation model** - describes how a user accumulates utility in the course of browsing $\Rightarrow E\left[g(D)\right] = \sum_{r=1}^{\infty} g(d) \cdot p(d)$

- Examples for the browsing models

  - *Position-based models* - the chance of observing a document depends on the position in the ranking. We can for example model it by $\Rightarrow p(d_r) = (1-\theta)^{r-1}\theta$. The corresponding utility accumulation is described by Rank-biased Precision (RBP): $RBP = \sum_{r=1}^{\infty} \text{rel}_r (1-\theta)^{r-1}\theta$
  - *Cascade-based models* - considers θ as a function of the document at rank r. Mostly, the following function is used: $\theta_r = \mathcal{R}(\text{rel}_r) = \frac{2^{\text{rel}_r}-1}{2^{\text{max rel}}}$. The corresponding utility accumulation is the *Expected Reciprocal Relevance*: $ERR@k = \sum_{r=1}^{k} \frac{1}{r} \cdot \theta_r \cdot \prod_{i=1}^{r-1} (1-\theta_i)$

TODO: add slides on choosing parameter values?

2.1.3 Novelty and diversity-based measures

- Assume that there is a distribution $P(i|Q)$ over intents for query $Q \rightarrow$ an intent-aware version of a measure is its weighted average over this distribution
- α-nDCG is generalization of nDCG that accounts for novelty and diversity
  - α is geometric penalty for redundancy
  - the gain of the document becomes:
    - +1 for each intent it is relevant To
    - $\times(1-\alpha)$ for each document higher in the ranking that intent already appeared in

2.1.4 Sessions and conversations

- So far focus on one-shot queries
- users frequently reformulate query $\rightarrow$ can we measure effectiveness of system over sequence of reformulations (session)? Can we optimize systems to provide better results over a session?
- User steps down ranked list of results until decision point and either abandons the search or reformulates
- while stepping down or sideways, the user accumulates utility
- **Session DCG:** From two queries with DCGs $\text{DCG}(RL1)$ and $\text{DCG}(RL2)$, we can compute the session DCG (does not consider early abandonment):

$$\text{SessionDCG} = \frac{1}{\log_c(1+c-1)}\text{DCG}(RL1) + \frac{1}{\log_c(2+c-1)}\text{DCG}(RL2)$$

- **Expected Global Utility:** Ω is space of all paths, $P(\omega)$ is prob of user following path $\omega \in \Omega$, $U(\omega)$ is utility of $\omega \rightarrow$ esM $= \sum_{\omega \in \Omega} P(\omega)U(\omega)$
  - User steps down ranking and stops based on stochastic process $\rightarrow$ reformulates query
  - $P(\omega) = P(r_1, r_2, \ldots, r_K)$ where r_i is stopping and reformulation point in ranking i
  - use geometric distribution to model stopping/reformulation behavior

2.1.5 Collection construction

- To evaluate a system offline, we need labels of whether a document is relevant with respect to a query (or graded scale) $\Rightarrow$ labels are created by humans
- First step is to generate a huge document collection, and generate a set of topics/queries that should be evaluated. Mostly, queries are selected from very frequent, common and rare query bin sets of highly-used search engines
- To be able to calculate measures like recall, we need to find all relevant documents in the collection. Can be done either deterministically or stochastically
  - **Depth-k pooling** - deterministic, standard method. Apply M IR systems and take the union of the k top results of all M systems. This set of documents is labeled by humans, and all others are considered as not relevant. Note that we need the M systems to be different/take another perspective on the data so that they don't find all the same documents. Otherwise, future IR algorithms can find relevant documents that the others haven't found yet and will be punished for that. k is task specific, but a value of 100 has shown to be sufficient
  - **Random Sampling** - stochastic method. Simplest approach is for a query q, just sample a small set of documents out of the whole corpus and label those. Otherwise are considered as unlabeled, thus neglected in evaluation. Problem: significant sparsity of relevant documents in the corpus.
  - **Stratified Random Sampling** Search engines bring relevant to the top -¿ oversample from top, undersample from bottom -¿ cannot take simple average but need to take weighted average

2.2 Challenges of offline evaluation

- Expensive and slow to collect new data
- Ambiguous queries are particularly hard to judge realistically (what intent is most popular?). Particularly hard for personalized searches
- Judges need to correctly appreciate uncertainty/allow different intents
- How to identify when relevance changes (temporal, query intent changes, ...)?

2.3 Comparative evaluation

- How do we compare different retrieval systems? Is the difference only due to random noise? $\Longrightarrow$ Statistical significance test
- Two hypotheses where we want to prove that H_0 is wrong:

$$H_0 : \text{MAP}_E - \text{MAP}_P = 0, \quad H_1 : \underbrace{\text{MAP}_E - \text{MAP}_P \neq 0}_{\text{two-sided}} \text{ or } \underbrace{\text{MAP}_E - \text{MAP}_P > 0}_{\text{one-sided}}$$

- Compute the p-value that describes the probability of observing the test data given that H_0 is valid (low p-value disprove null hypothesis)

2.4 Informativeness

- if you care about something, you should optimize for that
- this is not necessarily true if you don't have enough training data
- then, optimize for different metric that is however more informative for the algorithm
- an informative metric is a metric such that if I tell you the value, you can tell me the ranking for it
- e.g. given average precision value, infer probability of relevance of document at rank i: maximize entropy under the constraint that the average precision is at is

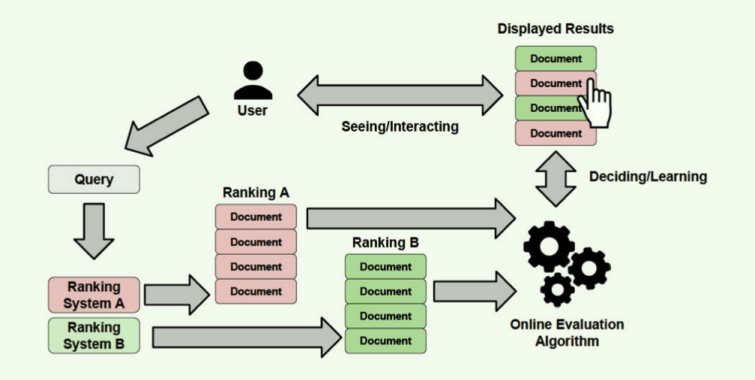


Figure 13: Schema of Online Evaluation

3 Online evaluation

- In online evaluation, the system interacts with the user $\implies$ user "tells" what is relevant (through interactions), system analyzes the user's behavior for gaining knowledge about preferences/relevances
- The benefit of online evaluations is that they are mostly simpler and more efficient (as they have control over what data is collected) and directly incorporate measuring the ranking quality
- online evaluation systems have control over what to display to user
- However, the downsides are that the results are worse to explain/interpret (why did users click less, different queries might rely on different metrics, ...). Also, evaluations might not be comparable over time so that we also need to ensure the same conditions/user population for both systems.
- An schematic figure is given in Fig. 13
- An online evaluation algorithm should
  - give reliable and correct results (robust to noise, handle biases)
  - provide good user experience

3.1 Analyzing user behavior

- A user provides various signals from which we can try to retrieve his "happiness" about the results. The following ones are mostly used:
- *Clicks* - clicks are mostly noisy so that a click doesn't ensure that the document was actually relevant. Clicks have several biases:
  - *Position bias* - a user tends towards clicking higher ranked results
  - *Contextual bias* - nearby results effect the click probability of a document
  - *Attention bias* - some results draw more attention to themselves by the usage of images, font size, ...
- **Time** - the time a user spends on a certain query before coming back to search engine
  - *Dwell time* - time spent on a clicked page. If duration is more than 30 seconds, we assume that click satisfied
  - *Exit type* - how the user exists the page (closing browser, continue scrolling through results, putting in new query, ...)
- *Mouse movement* - time on website is not sufficient. Mouse movement can indicate whether user is actually reading or only scrolling/scanning
- *Reformulations* - if new query is entered, check for similarity with the previous one. Reformulated/Similar queries that were entered quickly after the first one, indicate that user was not satisfied with previous results.

3.2 A/B Testing

- When testing two systems in an online experiment, we need to make sure that both have the same preconditions so that the system improvements clearly correlate with the new click/sale numbers
- In A/B Testing, users are split into two groups where each group is assigned to one of the algorithms. We analyze the users' behavior on both systems and calculate a metric based on that. Comparing the results for both systems on significance leads to a final decision.
- **Advantages**
  - straightforward and common
  - can test many different aspects
- *Challenges* in A/B Testing
  - If one system is very different and probably bad, it will affect the *user experience* and damages website image $\implies$ perform offline evaluation in advance to avoid testing a very bad system
  - It is hard to define *metrics* as they can contradict each other. For example, if we report number of clicks and sessions per users, a click increase can indicate better/more relevant results. However, if another systems provides snippets that already contain the information, the user will click less.
  - The metric should be as sensitive as possible. *Sensitivity* is the ability of the metric to detect the statistically significant difference when the treatment effect exists $\implies$ how many queries/days/users/... for significance needed?
  - they are inefficient and require lots of data
  - tests have to run for a long time
  - need to identify individual users

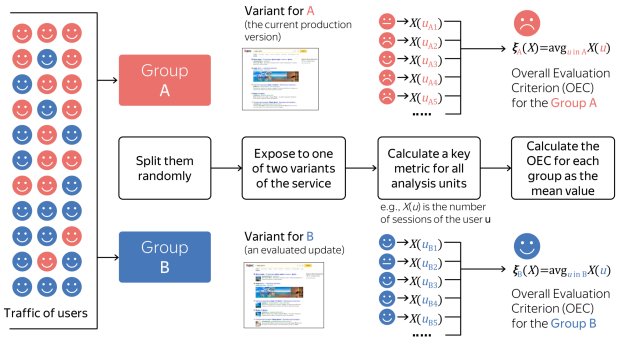


Figure 14: Visualization of A/B Testing

3.3 Interleaving

- A/B testing introduces a high variance by letting different users evaluate different systems $\implies$ Show interleaved results from both algorithms A and B without telling the user which document is from which model
- The evaluation is based on the clicks of a user where the algorithm gets the credit that provided the clicked document

3.3.1 Balanced interleaving

- In balanced interleaving, we select randomly which algorithm starts (A or B). If A would start, we take the first document of A and place it in our interleaved ranking list. Then we pick the first document of B and continue with A again
- If a document is already in the interleaved ranking, we skip this document and continue with picking the next document from the *other* ranking model

Algorithm 1 Balanced Interleaving

Input: Rankings $A = (a_1, a_2, ...)$ and $B = (b_1, b_2, ...)$
Init: $I = (); k_a \leftarrow 1; k_b \leftarrow 1$
$AFirst \leftarrow RandBit()$ *...decide which ranking gets priority*
while$(k_a \leq |A|) \wedge (k_b \leq |B|)$ **do** *...if not at end of A or B*
  if $(k_a < k_b) \vee ((k_a = k_b) \wedge AFirst = 1)$ **then**
    if $A[k_a] \notin I$ **then** $I \leftarrow I + A[k_a]$ *...append next A result*
    $k_a \leftarrow k_a + 1$
  else
    if $B[k_b] \notin I$ **then** $I \leftarrow I + B[k_b]$ *...append next B result*
    $k_b \leftarrow k_b + 1$
  end if
end while
Output: Interleaved ranking I

Figure 15: Formal algorithm describing balanced interleaving

- Problem: balanced interleaving brakes under corner cases. Assume following ranking:

$$A : \{d_1, d_2, d_3, d_4\}, B : \{d_2, d_3, d_4, d_1\}$$

  No matter whether we start at model A or B, the interleaved list contains three documents assigned to B and only one to A. Thus, random clicking would lead to B winning $\implies$ bias. Resolved by team-draft interleaving

3.3.2 Team-draft interleaving

- In team-draft interleaving we guarantee that both algorithms contribute equally to the interleaved ranking
- Process:
  1. Until k documents are placed:
     (a) randomly choose ranker A or B
     (b) let chosen ranker place its next unplaced document
     (c) let other ranker place its next unplaced document
     (d) remember which ranker placed which document
  2. present interleaving to user, observe clicks
  3. ranker with most clicks on its placed documents wins
- Team-draft interleaving finds no preference under random clicks (e.g. 50% chance that second document is clicked $\rightarrow$ equal for both A and B as they are equally likely to place second document)
- Properties of team-draft interleaving:
  - more efficient than A/B testing (needs less samples)
  - user experience hardly effectly (not worse than worst ranker)
  - correctness: ranker that placed clicked documents higher *usually* wins
- Problematic case for team-draft interleaving (document 3 being the only relevant document) in Fig. 16

3.3.3 Probabilistic interleaving

- To avoid biases completely, we can apply probabilistic models
- Convert the ranking of each model to a probability distribution by applying softmax ($\tau \in \mathbb{R}^+$) for ranker R_A:

$$P(d|D, R_A) = \frac{\frac{1}{rank(d,R_A)^\tau}}{\sum_{d' \in D} \frac{1}{rank(d',R_A)^\tau}}$$

- Process:
  1. compute P_A and P_B from ranker A and B
  2. repeat until k documents placed:
     (a) randomly choose P_A or P_B and sample document d
     (b) place d and remember which ranker placed it

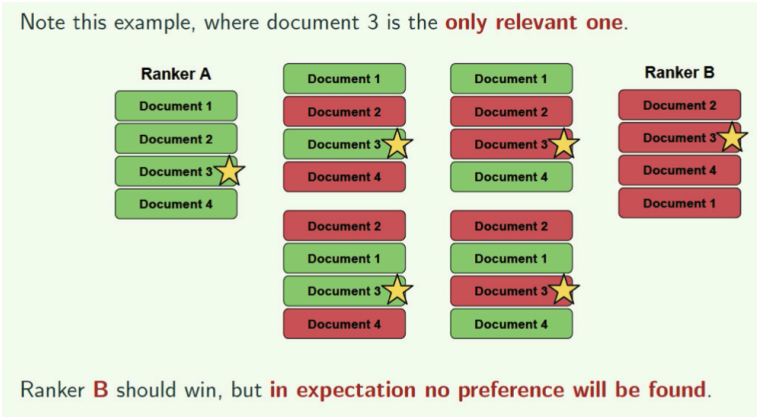


Figure 16: Edge Case of Team-Draft Interleaving

(c) renormalize P_A *and* P_B after removing d

3. display to user and observe clicks
4. ranker with the most clicked documents wins

- We expect the same number of clicks for documents at the same position of both algorithms due to the same probability in the softmax.
- Does this method have fidelity?
  - no ranker has advantage due to factors other than relevance as every ranker is equally likely to place at every rank
  - unambiguous winner will always win in expectation as every ranker is equally likely to place at every rank $\rightarrow$ dominating ranker is more likely to place relevant documents at each rank $\rightarrow$ if relevant documents are more likely to be clicked, dominating ranker wins in expectation

Probabilistic Interleaving with expected outcome

- Another, more efficient evaluation method is by marginalizing over all possible assignments. instead of using the outcome based on the ranker assignments, we calculate expected outcome over all possible ranker assignments
- let outcome of comparison be $O(R_A, R_B, L, T, c) \in \{-1, 0, 1\}$ where L is the interleaved list, assignments T, clicks c
- since clicks are independent of assignment T, we can marginalize:

$$\mathbb{E}[O(R_A, R_B, L, c)] = \sum_T P(T|R_A, R_B, L)O(R_A, R_B, L, T, c)$$

- the placement probability is therefore:

$$P(T_i = A) = 0.5$$

$$P(L_i = d|T_i = A) = P_A(d) = \frac{\frac{1}{rank(d, R_A)^\tau}}{\sum_{d' \in D} \frac{1}{rank(d', R_A)^\tau}}$$

$$P(T_i = A|L_i = d) = \frac{P_A(d)}{P_A(d) + P_B(d)}$$

- using that the outcome of a comparison is only dependent on the clicked documents and the assignment of a document does not depend on other assignments, we only have to consider the possible assignments of the *clicked* documents (bringing complexity from 2^k to 2^c)
- Process:

1. compute P_A and P_B from ranker A and B
2. repeat until k documents placed:

(a) randomly choose P_A or P_B and sample document d
(b) place d without remembering which ranker placed it
(c) renormalize P_A *and* P_B after removing d

3. display to user and observe clicks
4. calculate expected outcome by marginalizing over all possible placements
5. expected winner wins competition

- Note that compared to a hard assignment of 0 or 1 in balanced and team draft interleaving, the credit accumulated for clicks is smoothed based on the relative rank of the document in the original result lists (a click on any document leads to a non-zero credit for both rankings)
- Figure 17 visualizes an example for probabilistic interleaving

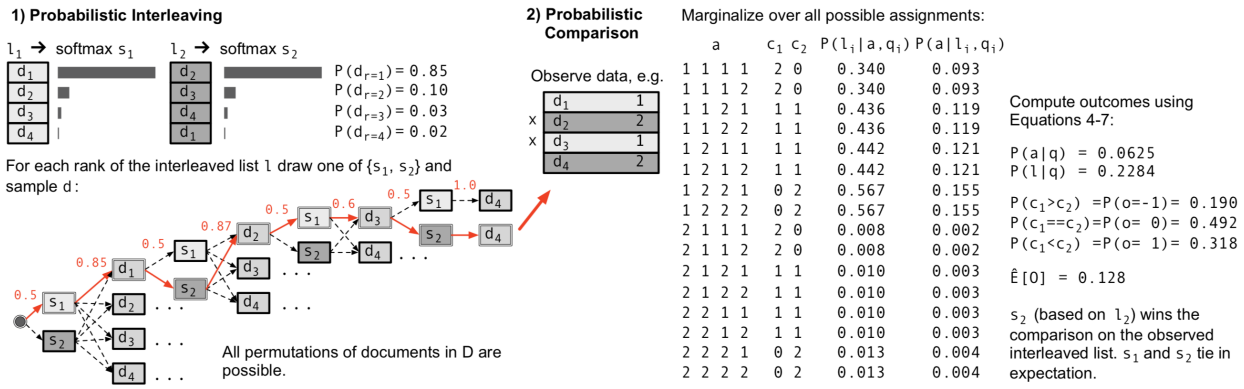


Figure 17: Visualization of probabilistic interleaving

Properties of probabilistic interleaving

- correctness:
  - correct outcomes guaranteed w.r.t. fidelity
  - marginalization does not affect expected outcomes
- User experience:
  - not guaranteed
  - every possible ranking can be displayed, albeit with different probabilities

3.3.4 Optimized interleaving

- look at allowed interleavings: produce rankings by always looking at the top documents, not yet placed $\rightarrow$ never have a document, where both rankers agree that another document should come first
- can use different scoring functions (there are requirements). A click on a document gives a preference score determined how document is ranked:
  - Linear Rank Difference:

$$s(d, R_A, R_B) = \delta_d = rank(d, R_A) - rank(d, R_B)$$

  - Inverse Rank Difference:

$$s(d, R_A, R_B) = \delta_d = \frac{1}{rank(d, R_A)} - \frac{1}{rank(d, R_B)}$$

- This gives us Fig 18 where, if the probability p_L of interleaving L being displayed, the expected outcome can be written as:

$$\mathbb{E}[O] = \sum_{L \in \mathcal{L}} \left(p_L \sum_{i=1}^{|L|} P(click(i)) s(L_i, R_A, R_B) \right) = 0$$

which becomes a linear optimization task to find p_L

Given two rankers where $R_A = [1, 2, 3, 4]$ and $R_B = [2, 4, 3, 1]$ and using the *Linear Rank Difference* scoring function, a possible solution is:

Interleaving	δ_{L_1}	δ_{L_2}	δ_{L_3}	δ_{L_4}	$E[O]$	p_L
$[1, 2, 3, 4]$	3	-1	0	-2	$3P(click(1)) - P(click(2)) - 2P(click(4))$	0%
$[1, 2, 4, 3]$	3	-1	-2	0	$3P(click(1)) - P(click(2)) - 2P(click(3))$	25%
$[2, 1, 3, 4]$	-1	3	0	-2	$-P(click(1)) + 3P(click(2)) - 2P(click(4))$	0%
$[2, 1, 4, 3]$	-1	3	-2	0	$-P(click(1)) + 3P(click(2)) - 2P(click(3))$	35%
$[2, 4, 1, 3]$	-1	-2	3	0	$-P(click(1)) - 2P(click(2)) + 3P(click(3))$	40%
$[2, 4, 3, 1]$	-1	-2	0	3	$-P(click(1)) - 2P(click(2)) + 3P(click(4))$	0%

Figure 18: Optimized Interleaving

- We used randomization to check that under randomized clicks there is no winner
- Properties of optimized interleaving:
  - user experience: strongest guarantees of all interleaving methods
  - correctness: method has fidelity (if optimized for it) as long as linear optimization is successful (it has been proven by brute force that there always is a solution for top-10 rankings)
  - can be correct under other definitions as well

4 Click models

- User clicks can be used as evaluation of IR systems as clicks indicate the relevance of a document
- However, clicks are highly biased (positional, textual, attention/visual,...) $\Rightarrow$ click models try to remove these biases and help using clicks for evaluation
- Click models are optimized/trained on click logs which record for a given query which documents were clicked
- Most models are based on probabilistic graphical models (PGMs) that describe the probability of a click
- They are mostly trained by either applying a MLE or EM algorithm

4.1 Random click model

- In random click models, every document on the result page has the same probability of being clicked:

$$P(C_u = 1) = \text{const} = \rho$$

- Therefore, the model contains only a single parameter, which can be optimized by applying MLE:

$$\rho = \frac{\#\text{clicks}}{\#\text{shown docs}}$$

- *Advantages*: simple and fast
- *Disadvantages*: the random click model does not consider many aspects including the position and content of a document
- There are different variations of this model (also called click-through rate models - CTR) considering more aspects
  - **Rank-based CTR** - modeling a probability for every rank on the result page: $P(C_{u_r} = 1) = \rho_r$
  - **Query-document CTR** - modeling a probability for every query-document pair in the dataset: $P(C_u = 1) = \rho_{uq}$

4.2 Position-based model

- Position-based models take the position *and* the document-query pair into account for modeling the probability of a click
  - *Examination* - reading a snippet at a rank/position $\implies P(E_r = 1) = \gamma_r$
  - *Attractiveness* - prob. for document-query relevance $\implies P(A_{uq} = 1) = \alpha_{uq}$
  - The combined probability of clicking on a document is therefore:

$$P(C_u = 1) = P(E_{r_u} = 1) \cdot P(A_{uq} = 1)$$

- The model is visualized in Figure 19

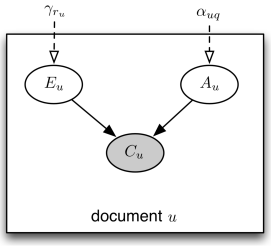


Figure 19: Probabilistic graphical model of parameters for PBM

- The examination models the position bias in user clicks while the attractiveness covers the document relevance
- *Advantages*: Distinguishing between position bias and document relevance
- *Disadvantages*: the Position-based model assumes that all clicks are independent of each other. Models that overcome this include:
  - *User browsing model (UBM)* - examination is also based on the rank of the previously clicked document $\implies P(E_{r,r'} = 1) = \gamma_{r,r'}$ ($n + n \cdot (n-1)/2$ parameters $\rightarrow$ 55 parameters for $n = 10$)
  - *Cascade model* - see next section

4.3 Cascade model

- The cascade model assumes that the user scans the documents from top to bottom until he finds a relevant document and clicks
- Thus, the top document is always examined, while following documents are only examined if none of the previous ones were clicked
- The cascade model can be summarized in the equations:

$$\begin{aligned}
P(A_r = 1) &= \alpha_{u_r q} \\
P(E_1 = 1) &= 1 && \textit{first element is always examined} \\
P(E_r = 1 | C_{r-1} = 1) &= 0 && \textit{stop if previous document is clicked} \\
P(E_r = 1 | E_{r-1} = 0) &= 0 && \textit{only examine if none of the documents before was clicked} \\
P(E_r = 1 | E_{r-1} = 1, C_{r-1} = 0) &= 1 && \textit{if no click was performed yet, examine next document}
\end{aligned}$$

- Therefore, the model has no parameters for examination and solely relies on attractiveness. The corresponding PGM is visualized in Figure 20
- *Advantages*: Clicking on a document depends on previous decisions/documents
- *Disadvantages*: No skips are allowed. Also, the cascade model only considers a single click $\implies$ Dynamic Bayesian Networks

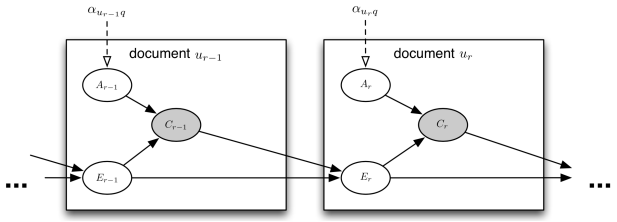


Figure 20: Probabilistic graphical model of parameters for CM

4.4 Estimation in Click models

- Cascade-based models only have one type of parameter (α) that do not depend on one another $\rightarrow$ use maximum likelihood
- In position-based models we cannot estimate examination and attractiveness separately $\rightarrow$ use EM algorithm
  - Estimate α_{uq} by using a click-log for a query-document pair $\mathcal{S}_{qd}$:

$$\alpha_{uq}^{(t+1)} = \frac{1}{|\mathcal{S}_{qd}|} \sum_{s \in \mathcal{S}_{qd}} \left(c_d^{(s)} + (1 - c_d^{(s)}) \frac{(1 - \gamma_r^{(t)}) \alpha_{qd}^{(t)}}{1 - \gamma_r^{(t)} \alpha_{qd}^{(t)}} \right)$$

  - Estimate γ_r also by using a click-log $\mathcal{S}$:

$$\gamma_r^{(t+1)} = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \left(c_d^{(s)} + (1 - c_d^{(s)}) \frac{\gamma_r^{(t)} (1 - \alpha_{qd}^{(t)})}{1 - \gamma_r^{(t)} \alpha_{qd}^{(t)}} \right)$$

4.5 Applications of click models

- Full click probabilities (i.e. probability of a click, no matter what) can inform model-based metrics (e.g. Rank-based precision and expected reciprocal rank)
  - Utility-based metrics: $\text{uMetric} = \sum_{r=1}^{n} P(C_r = 1) \cdot U_r$
  - Effort-based metrics (how much effort it takes to examine/perceive document at position): $\text{eMetric} = \sum_{r=1}^{n} P(S_r = 1) \cdot F_r$ ($P(S_r = 1)$ is probability of success/satisfaction)
  - From reciprocal rank $RR = \frac{1}{r}$, where $S_r = 1$ we get expected reciprocal rank $ERR = \sum_r \frac{1}{r} \cdot P(S_r = 1)$
    - Can be estimated using:

$$\begin{aligned} P(A_r = 1) &= \alpha_{qd_r} \\ P(E_1 = 1) &= 1 \\ P(E_r = 1 | S_{r-1} = 1) &= 0 \\ P(E_r = 1 | S_{r-1} = 0) &= \gamma \\ P(S_r = 1 | C_r = 0) &= 0 \\ P(S_r = 1 | C_r = 1) &= \sigma_{qd_r} \end{aligned}$$

    - then:

$$\begin{aligned} P(S_r = 1) &= P(S_r = 1 | C_r = 1) \cdot P(C_r = 1) \\ &= \sigma_{qd_r} \cdot \alpha_{qd_r} \cdot \prod_{i=1}^{r-1} (\gamma \cdot (1 - \sigma_{qd_i} \cdot \alpha_{qd_i})) \\ &= R_{qd_r} \cdot \prod_{i=1}^{r-1} (\gamma \cdot (1 - R_{qd_i})) \end{aligned}$$

    - putting into ERR:

$$ERR = \sum_r \frac{1}{r} \cdot R_{qd_r} \cdot \prod_{i=1}^{r-1} (\gamma \cdot (1 - R_{qd_i}))$$

- Conditional click probabilities (i.e. probability of click given previous clicks) can be used to simulate users/clicks (go to position 1, get click probability and examine/go to next document)
- Click models also give parameter values (α and γ) which are unbiased and can be used for (learning to) ranking: e.g. in counterfactual LTR the examination probabilities can be used to estimate the observation probabilities (which are used for de-biasing in IPS/IPW) or the α for labels in LTR

5 Introduction to Retrieval models

- Mathematical framework for defining query-document matching

5.1 Query Analysis

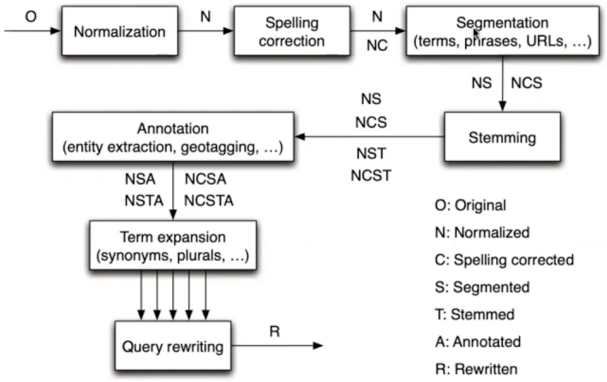


Figure 21: Query analysis pipeline

Query analysis pipeline is given in Fig. 21.

5.2 Query Processing

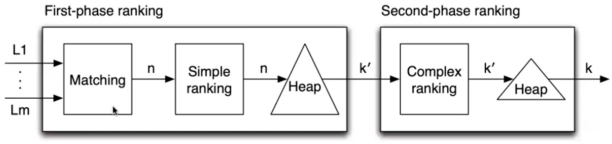


Figure 22: Query processing

- Process depicted in Fig. 22
- Matching: select documents that have something to do with query
- Simple ranking: retrieve first ranking for a large number of documents
- Complex ranking: re-rank only top-k documents from simple ranking (simple ranking is fast and cheap whereas complex ranking is slow and expensive)

How to select documents?

- Term-at-a-time: preprocess terms $\rightarrow$ for each term, go through inverted list of documents (and weights) $\rightarrow$ add weights of corresponding documents across terms (advantages: only one list at a time to process, disadvantages: store complete list of weights in memory)
- Process all inverted lists at the same time $\rightarrow$ like merge sort where you keep pointers in each list and have those pointers at the same document ids all the time (note that inverted lists have to be ordered)

- In practice: for websearch select documents where ALL query terms appear in documents; in cases where you want to retrieve many possible matches, select documents where at least one term from query appear (e.g. medical search)
- A document score is usually computed as linear combination of query-dependent and query-independent scores

5.3 TF-IDF

- In a vector space model, documents and queries are represented in vector space
- Axes are mostly terms/vocabulary so that a document or query is represented by terms they contain (binary) or their frequency
- We can rank documents based on their cosine similarity with the query:

$$\text{score}(d, q) = \frac{\vec{q} \cdot \vec{d}}{||\vec{q}|| \cdot ||\vec{d}||}$$

- Documents can be therefore represented as non-negative vector of term weights (raw frequency in doc):

	nuclear	weapon	bomb	stockpile	chemical	power	reactor	the
doc1	252	182	58	5	7	12	5	245
doc2	423	18	0	0	0	236	160	365
doc3	0	94	0	38	110	0	0	85
query	1	0	0	1	0	0	0	1

- However, the problem here is that terms with a higher frequency in documents are automatically more important, although this is not always the case (e.g. ”the”). Thus, for identifying the important terms, we can report document frequency (no. of docs in which terms occurs):

$$\text{df}(t) \coloneqq \#\{d : \text{tf}(t; d) > 0\}$$

- We can translate document frequencies to term weights by inverting them (inverted document frequency - *IDF*):

$$\text{idf}(t) = \log \frac{n}{\text{df}(t)} = \log n - \log \text{df}(t)$$

  The log is applied to dampen the effect of IDF.

- Also the term frequencies should be dampened by a monotonic, sub-linear transformation as a term occurring twice as often doesn't imply that the document is also twice as important/relevant. Together, we can define the tf-idf weights as follows:

$$\text{tf-idf}(t; d) = \log(1 + \text{tf}(t; d)) \log \frac{n}{\text{df}(t)}$$

- Scores are normalized by euclidean distance of document. Alternatively, we could also apply tf-idf on the relative term frequencies.

5.4 BM25

- Probabilistic retrieval framework that extends the idea of tf-idf
- Instead of the log, we use a different damping functions which are easier to control:

$$w_t = \frac{(k_1 + 1) \cdot \text{tf}(d; t)}{k_1 + \text{tf}(d; t)} \cdot \text{idf}(t)$$

- In addition, we normalize the term frequency by the document length: $\text{tf}'(d; t) = \text{tf}(d; t) \cdot l_{avg}/l_d$ (l_{avg} is the average document length of collection). By this we prevent copies of documents concatenated with each other being higher rated. Putting this into our original function, we get:

$$w_t = \frac{(k_1 + 1) \cdot \text{tf}(d; t)}{k_1 \cdot (l_d/l_{avg}) + \text{tf}(d; t)} \cdot \text{idf}(t)$$

- However, longer documents also tend to contain more information. Thus, we introduce another parameter b that controls the normalization:

$$w_t = \frac{(k_1+1)\cdot \text{tf}(d;t)}{k_1\cdot((1-b)+b\cdot(l_d/l_{avg}))+\text{tf}(d;t)}\cdot \text{idf}(t)$$

- For very long queries, we also need to consider this normalization which can be done by multiplying another term $\frac{(k_3+1)\cdot \text{tf}(q;t)}{k_3\cdot \text{tf}(q;t)}$
- In conclusion, the BM25 score is calculated as follows:

$$\text{BM25} = \sum_{\text{unique } t\in q} \frac{(k_1+1)\cdot \text{tf}(d;t)}{k_1\cdot((1-b)+b\cdot(l_d/l_{avg}))+\text{tf}(d;t)}\cdot\frac{(k_3+1)\cdot \text{tf}(q;t)}{k_3+\text{tf}(q;t)}\cdot \text{idf}(t)$$

- Parameters k_1, b and k_3 are tuned. Common defaults are $k_1 = 1.5$ and $b = 0.75$
- It is the most widely used ranking in IR but only loosely inspired by probabilistic models
- Parameters:
  - $k_1 \to 0 \Rightarrow$ Sum of idf
  - $k_1 \to \infty \Rightarrow$ Sum of tf-idf
  - b is parameter controlling the document length, if $b = 0 \Rightarrow$ no normalization by document length $\Rightarrow$ not fair against short documents
  - if $tf \to \infty \Rightarrow$ reduces to sum of idf with constant multiplication $(k_1 + 1)$
  - long queries: cancel out with large term frequency in query

5.5 Statistical Language Models

- Statistical language models are a probability distribution over word sequences $P(w_1, ..., w_m)$ with which documents and queries can be represented (and uncertainty quantified)
- Thus, a language model describes the probability of e.g. q being the given word sequence
- Documents are ranked given a query. We can use either document likelihood, query likelihood or KL-divergence

5.5.1 Query likelihood

- Given a document, what queries are most likely to be created for it?
- We first have to ensure that the query likelihood correlates with document likelihood. Therefore, we apply the Bayes rule: $p(d|q) = \frac{p(q|d)p(d)}{p(q)}$. As $p(q)$ is equal for all documents, and we assume a uniform prior for all documents (though not always the case), we retrieve $p(d|q) \propto p(q|d) = \prod_{t\in q} p(t|d) = \prod_{t\in q} \frac{tf(t,d)}{dl(d)}$
- Thus, by generating a probability distribution of possible queries for a document, we can approximate how likely a document is given a query.
- The scoring function is defined as follows:

$$\text{score}(d, q) = \log\left[p(q|\theta_d)\cdot p(d)\right]$$

  where θ_d describes the document.

5.5.2 KL-Divergence

$$KL(\theta_d||\theta_q) = \sum_{t\in V} p(t|\theta_q)\log\frac{p(t|\theta_q)}{p(t|\theta_d)}$$

5.5.3 Smoothing

- How to deal with unseen words which have a probability of 0.
- First, we assume a multinomial distribution again with the optimal parameters of $p(t|\theta_d) = \frac{\text{tf}(t,d)}{dl(d)}$
- **Adaptive smoothing**: add a small extra count to every word:

$$p(t|\theta_d) = \frac{\text{tf}(t,d) + \epsilon}{dl(d) + \epsilon|V|}$$

  In case of $\epsilon = 0$, we fall back to ML estimation. $\epsilon = 1$ is called Laplace smoothing.

- **Jelinek-Mercer smoothing**: linearly interpolate with "background" knowledge so that rare words also have smaller additives:

$$p_\lambda(t|\theta_d) = \lambda \frac{\text{tf}(t,d)}{dl(d)} + (1-\lambda)\frac{\text{cf}(t)}{|C|}$$

  where C is the whole collection and cf is the collection frequency.

- **Dirichlet prior smoothing**: we assume that before seeing the document, we have a prior belief over all words in it $p(\theta_d)$. We use the posterior which gets narrower the more words we see and therefore the more certain we are about the document distribution.
  - Maximum A Posteriori estimate by $\hat{\theta}_d = \arg\max_{\theta_d} p(\theta_d|d) = \arg\max_{\theta_d} p(d|\theta_d)p(\theta_d)$
  - Prior distribution $p_i \sim \text{Dir}(\alpha) \implies p(\theta_d) = \prod_{w \in V} p(w|\theta_d)^{\alpha_w - 1}$
  - With a multinomial likelihood, we get:

$$p(\theta_d|d) \propto \prod_{w \in V} p(w|\theta_d)^{\text{tf}(w;d)} \prod_{w \in V} p(w|\theta_d)^{\alpha_w - 1} = \prod_{w \in V} p(w|\theta_d)^{\text{tf}(w;d) + \alpha_w - 1}$$

  - Thus, our new MAP solution is:

$$p(w|\theta_d) = \frac{\text{tf}(w;d) + \alpha_w - 1}{|d| + \sum_{w \in V} \alpha_w - |V|}$$

  - For $\alpha_w = 1$, we get MLE estimation, and $\alpha_w = 2$ represents Laplace smoothing.
  - We can also rewrite the smoothing similar to Jelinek-Mercer smoothing:

$$p(w|\theta_d) = \frac{|d|}{|d| + \mu}\frac{\text{tf}(w;d)}{|d|} + \frac{\mu}{\mu + |d|}p(w|C)$$

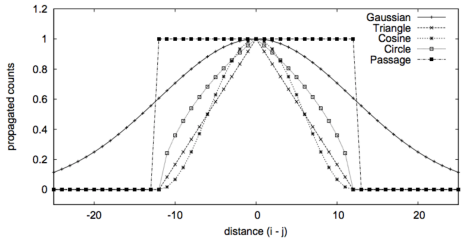
    where μ is the parameter depending on α_w. Thus, we interpolate with the background knowledge while taking the document length into account.

- Next to Dirichlet prior smoothing, we can also use other distributions (for example a beta prior with multiple Bernoulli) which lead to slightly different smoothing functions. For example, with the beta prior, we get for a variable α_w and β_w (without constraints!):

$$p(w|\theta_d) = \frac{\text{tf}(w;d) + \alpha_w - 1}{\alpha_w + \beta_w - 1}$$

5.5.4 Positional Language Models

- There are variants of basic language models capturing term dependencies
- Instead of having one language model representing the whole document, Positional Language Models define a LM for every word position
- Thus we capture (small) "fuzzy" passages with which we can match our query
- A term at each position can propagate its occurrence to close positions in word windows
  - Example sentence: `the black hat is not`...
  - With a equally weighted word window of one, we would retrieve the following language model (MLE params) for the position of word "`black`": $p(\texttt{black}|\theta_p) = 1/3,\ \ p(\texttt{the}|\theta_p) = 1/3,\ \ p(\texttt{hat}|\theta_p) = 1/3$



- We can weight the occurrences of every word based on the distance to the "root" of the language model (also called kernel):
- In general, the term frequency of a word for a LM at position j with kernel k is determined as follows:

$$\text{tf'}(w, j; d) = \sum_{i=1}^{|d|} \text{tf}(w, i; d) \cdot k(i, j)$$

- The language model at every position is given by the corresponding MLE estimation:

$$p(w|d, j) = \frac{\text{tf'}(w, j; d)}{\sum_{w' \in V} \text{tf'}(w', j; d)}$$

- Documents can now be scored by either their best matching language model with the query, or the average of the top-k models

6 Semantic matching

- *Vocabulary gap*: query and document might use different lexical representation for the same entity $\implies$ resolve by semantic matching
- Represent query and document by their meaning, not lexical/word level
- This will help to identify synonyms and/or semantic relatedness for computing similarity
- To get such a representation, one idea is to apply dimensionality reduction. This relies on the assumption that the dimension of the data is actually lower than i.e. vocabulary size
- Similar data in terms of semantic will be (hopefully) similar in reduced dimensions as well

6.1 Latent Semantic Indexing

- We represent all documents and queries in a term-document matrix:

$$X = \begin{bmatrix} | & | & & | \\ x_1 & x_2 & \dots & x_m \\ | & | & & | \end{bmatrix}$$

  where m columns represent the documents, and n rows the terms. A single cell in the matrix X specifies the term frequency $\text{tf}(w; d)$. Note that we would also add the queries in X as documents.

- On this matrix, we apply Singular Value Decomposition (SVD) so that $X_{n \times m} = U_{n \times r} \Sigma_{r \times r} V^T_{m \times r}$
  - $\boldsymbol{U}_{n \times r}$ represents the word/term embedding along rows (one word per row). The columns are the "semantic" dimensions that e.g. represent topics/hidden lower-dimensional space.
  - $\boldsymbol{V}_{m \times r}$ similarly represents the embedding of documents (one doc per row, but is transposed in calculation).
  - $\boldsymbol{\Sigma}_{r \times r}$ is a square, diagonal matrix. The magnitude of a singular value represents the importance of the corresponding latent dimension in the collection/data. It is always sorted from the highest value in first place to lowest value in last place.

- To reduce dimensions to k, we simply drop those with the lowest values in Σ. These dimensions may be noise and make things dissimilar when they actually are on topic level. k is hyperparameter.

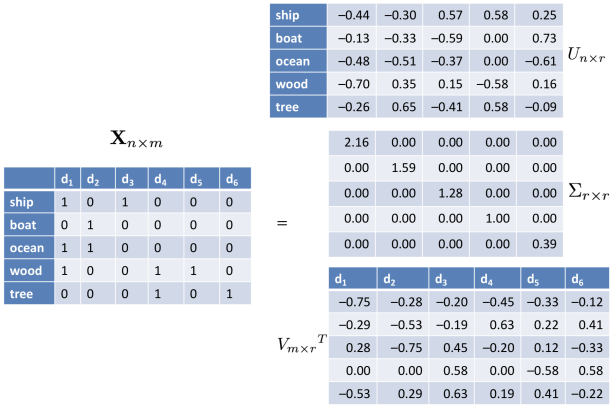


Figure 23: Example of SVD for 6 documents and 5 terms

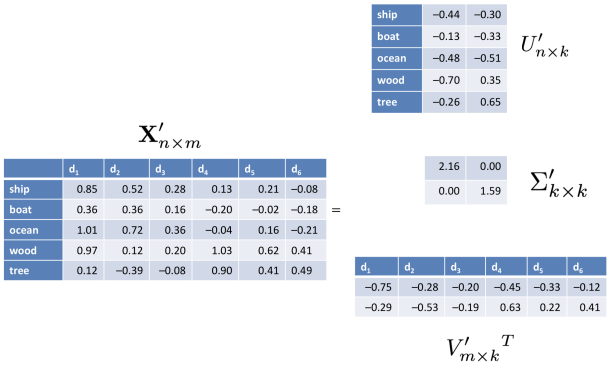


Figure 24: Reduced dimensions by $k = 2$ for 6 documents and 5 terms

- In case of Figure 23 with $k = 2$, we would drop the last three dimensions. This leads to our new embeddings X' for the documents. The resulting matrices would look like as in Figure 24.
- document d and query q (containing binary scores) can be encoded by:

$$\hat{d} = \Sigma_k^{-1} U_k^T d$$
$$\hat{q} = \Sigma_k^{-1} U_k^T q$$

- The similarity between documents/queries can be calculated by cosine similarity (dot product) between their new embeddings in X'. Furthermore, we can also compute the similarity between terms by using the rows.
- Choice of k:
  - The choice of k is critical in IR. The ideal value of k would be large enough to fit all the real structure in the data, but small enough to compress/group terms together that are very similar (less noise).
  - Typically, different values of k are tested and compared by their performance. For example, a high precision but low recall suggests a poor generalization of the model. Therefore, we should decrease k in this case.
- LSI addresses synonymy by mapping similar words in the same dimensions. The cost of such a mapping is lower than for unrelated words as they occur similar/same documents.
- **Strengths** of LSI
  - Using X' instead of X show performance increase as we filter out the noise
  - X' represents the best approximation of X with a matrix of rank k: $X' = \underset{X':\text{rank}(X')=k}{\arg\min} ||X - X'||$
  - Is mostly combined with lexical methods like BM25 to not lose "obvious" matches

- **Weaknesses** of LSI
  - A huge storage is required as the matrices U and V are dense (less zeros)
  - Representations are not interpretable, and it is not guaranteed that hidden dimensions represent topics
  - k is often not easy to determine and requires multiple tests
  - SVD assumes orthogonal dimensions on which the variance is maximum which is not always the case
  - The model is not generative or probabilistic, which makes it hard to extend collection by new documents/queries (worst case: redo whole SVD)
- One alternative is Non-negative Matrix Factorization which leads to smaller, positive matrices (but doesn't solve the other problems)

6.2 Probabilistic latent semantic analysis (pLSA)

- (Pseudo-)generative model with which we try to detect key topics in the collection in an unsupervised fashion. The model describes how we would generate docs for certain topics
- Every topic has its own language model/distribution over words in vocabulary in a unigram/bag-of-word style: $p(w|z = 1), p(w|z = 2), ...$ where z is the variable representing the topics.
- A document is represented by the distribution of these topics in the document. Generating a document would require to first sample a topic based on the topic distribution of the document, and then generating a word based on the language model of the topic. The order of the words is not taken into account. An example is shown in Figure 25.

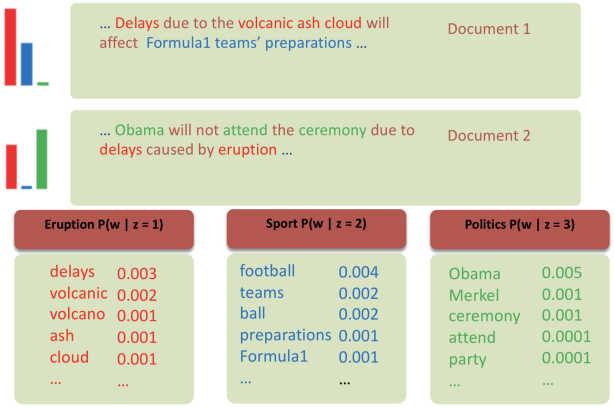


Figure 25: Example of document representation in pLSA

- For semantic matching, we calculate the KL divergence between the topic distributions of the query and the document or use the query likelihood of either document given query or query given document.
- pLSA can have problems with stop words (common words that occur in every document frequently like "the" or "and"). To prevent that, we summarize all stop words in a separate background topic model which is equally shared by all docs. Before generating a word, we toss a biased coin to decide whether to retrieve a word from the background or standard topic model. Note that in matching, these stop words are masked out by that

6.2.1 Retrieving distributions

- Input: collection of N documents, number of topics K
- Output:
  - Distributions over words $\phi_{(z,w)} = p(w|z)$ for $z \in \{1, ..., K\}$ with $\sum_{w \in V} p(w|z) = 1$
  - Distributions over topics in all documents: $\theta_{d,z} = p(z|d)$ for every d, with $\sum_{z=1}^{K} p(z|d) = 1$

- We try to solve problem by MLE. The probability of w appearing at position i in the document d is:

$$p(d_i = w|\Phi, \theta_d) = \sum_{z=1}^{K} \phi_{(z,w)}\theta_{d,z}$$

  The joint likelihood of the entire dataset is:

$$p(W|\Phi, \Theta) = \prod_{d\in D}\prod_{w\in V}\left(\sum_{z=1}^{K}\phi_{(z,w)}\theta_{d,z}\right)^{\text{tf}(w;d)}$$

- Taking the two constraints into account, we get an optimization problem which we can solve by the EM algorithm. For that, we assume that we know from which topic a word was generated at position i in the document by R_{d_i}:

$$p(W|R, \Phi, \Theta) = \prod_{d\in D}\prod_{i=1}^{N_i}\sum_{z=1}^{K} R_{(d_i,z)}\left(\phi_{(z,w)}\theta_{d,z}\right)$$

- For the EM algorithm, we would update R_{d_i} during the expectation step by $R_{d_i} = \frac{\phi_{(z,w_i)}\theta_{(d,z)}}{\sum_{z=1}^{K}\phi_{(z,w_i)}\theta_{(d,z)}}$. The maximization step consists of updating Θ and Φ: $\theta_{(d,z)} = \frac{\sum_{d_i} R_{(d_i,z)}}{|d|}$, $\phi_{(z,w)} = \frac{\sum_{d\in D} n(d,w)R_{(w,z)}}{\sum_{w'\in V}\sum_{d\in D} n(d,w')R_{(w',z)}}$
- PLSA is able to learn topics with their corresponding word distributions. However, there are still some drawbacks:
  - It is still not a fully generative model. After running PSLA, we have topic distribution for documents we initially had, but we cannot extend it to new documents (or only hardly with heuristics)
  - Prone to overfitting

6.2.2 Latent Dirichlet Allocation (LDA)

- Takes PLSA but makes it a generative model with Dirichlet prior. Can also be seen as a Bayesian treatment of PLSA
- Instead of probabilities for every document, we now simply define two hyperparameters α and β
- For every topic $z = 1, ..., K$, we draw a word distribution $\phi_z \sim \text{Dir}(\beta)$. Thus, β determines how words are distributed per topic
- For each document d, we sample a topic distribution $\theta_d \sim \text{Dir}(\alpha)$. The alpha therefore controls the mixture of topics for any given document.
- The words in the documents are then sampled by the probabilities ϕ_z and θ_d. Note that this is a fully generative model as we can generate as many new documents as we need.

6.2.3 Graphical Models and Probabilistic Topic Models

- We can represent every probabilistic topic model as graphical model which abstracts the conditional independence relationships
- For example, Figure 26 visualizes the graphical model of the Latent Dirichlet Allocation

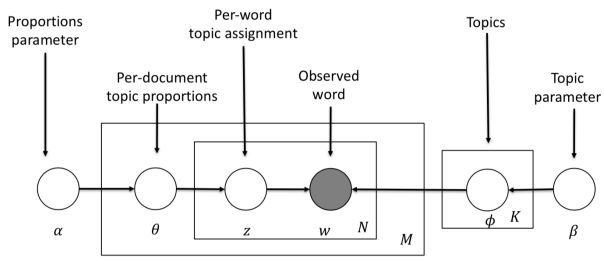


Figure 26: Graphical model of LDA

- In this diagrams, it is easier to show extensions. For instance, Figure 27 visualizes the Author-Topic model where we add an observed variable of the author of a document. This observation affects the probability distributions over words for each topic in a document

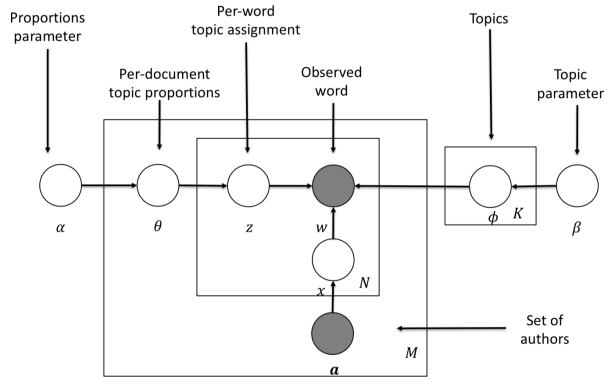


Figure 27: Graphical model of the Author-topic model

7 Neural Retrieval Models

7.1 Distributed Word Representations

- Latent, dense vector representation to model semantic similarity/relations

7.1.1 Skip-gram

- **Skip gram**: learn to predict neighboring words in a small context window
- Model probability by similarity between word and context vectors (two matrices):

$$p(w_k|w_j) = \frac{\exp(c_k \cdot v_j)}{\sum_{i \in |V|} \exp(c_i \cdot v_j)}$$

- Denominator can be computationally expensive if vocabulary is quite large. Thus, we can approximate it by taking just a few negative examples $\implies$ negative sampling
- Overall, skip gram will learn two representations for each word (context C and target words V) from which me most likely only use V (visualized in Figure 28)

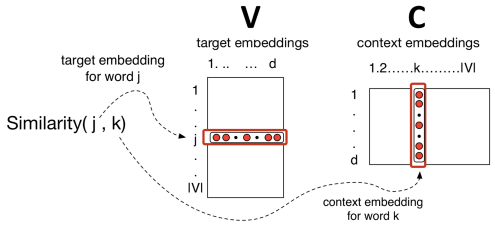


Figure 28: Visualization of skip gram method for learning word representations

- Skip gram shows to capture relational meaning (`KING - MAN + WOMAN = QUEEN`)

7.2 Compositionality

- To match queries and documents in the embedding space, we need to combine the words in each $\implies$ compositionality

7.2.1 Aggregate word vectors

- Apply simple rules/arithmetic to combine word vectors
- Example: **Dual Embedding Space Model**
  - represent a document by the centroid of its word vectors $\overline{\boldsymbol{D}} = \frac{1}{|D|} \sum_{\boldsymbol{d}_j \in D} \frac{\boldsymbol{d}_j}{||\boldsymbol{d}_j||}$

– The query-document similarity is the average over query words of cosine similarity: $\text{DESM}(Q, D) = \frac{1}{|Q|} \sum_{q_i \in Q} \frac{\boldsymbol{q}_i^T \overline{\boldsymbol{D}}}{||\boldsymbol{q}_i|| \cdot ||\overline{\boldsymbol{D}}||}$
– We can also use both the IN (word) and OUT (context) embeddings from skip-gram to optimize matching. What worked best was using IN representations for the query and OUT for document
– In the ranking system, we either first rank documents by BM25 and rerank top N with DESM, or use a linear combination of both scores

7.2.2 Tune and Aggregate word vectors

- Learn task-specific representations and not rely on pure skip-gram
- **Paragraph2vec**
  – Generalizes word2vec to whole documents by embedding them in a fixed-size vector
  – Two different approaches. First is *distributed memory*:
    * We are trying to predict the next word based on a few previous context word *and* a paragraph embedding.
    * Both the word and paragraph embeddings are learned during this process
    * Input embeddings can either be concatenated or averaged (commonly first one is applied)
    * Visualization in Figure 29

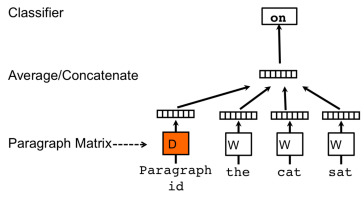


Figure 29: Distributed memory model.

  – Second method: *Distributed bag of words*
    * In this approach, we don't consider the context words but try to predict all possible words in the paragraph given the embedding vector
    * This is done by sampling a random word at every SGD iteration from the small text windows, and train the classifier on predicting this word
    * Thus, we optimize the embedding regarding representing the word distribution in the paragraph
    * The distributed BOW is visualized in Figure 30

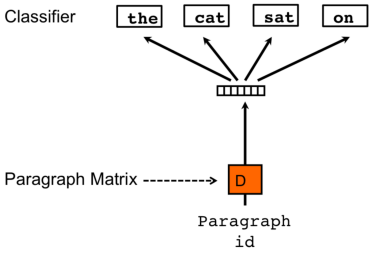


Figure 30: Distributed BOW model.

  – When having new query, no embedding vector exists yet
  – Given a query
    * Fix the word matrix W
    * Add a (randomly initialized) column to the document matrix D corresponding to the query representation
    * Update D using gradient descent
    * Get the vector representation of the query from the updated matrix D

8 Learning to Rank

- Main issue in information retrieval is to determine whether document d is relevant for query q
- Common relevance signals include TF-IDF, BM25, document popularity, page rank, spam identifier etc.
- But: what signals to use/how to combine these signals? There is not a single relevance signal "to rule them all" $\implies$ combine all signals in a model
- Simplest combination method: linear model $f(\boldsymbol{d}, \boldsymbol{\theta}) = \sum_{i=1}^{|d|} \theta_i d_i$ where $\boldsymbol{d}$ represents the different signals for document-query pair
- Task: find the optimal parameter set $\boldsymbol{\theta}$, commonly by Machine Learning techniques (linear regression)

8.1 Offline Learning To Rank

- Given an annotated dataset of relation document and relevance/ranking
- expensive and time-consuming
- provides ground-truth
- There are three different approaches: pointwise, pairwise, listwise

8.1.1 Pointwise

- optimize models $f(\boldsymbol{d}, \boldsymbol{\theta})$ to predict relevancy of a single document. This can be recasted in a regression problem with loss:
$$\mathcal{L} = \sum_{\boldsymbol{d}} \left(f(\boldsymbol{d}, \boldsymbol{\theta}) - \text{relevancy}(d, q)\right)^2$$
- Could also use classification approach to predict rank of document
$$\mathcal{L} = \sum_{\boldsymbol{q}, \boldsymbol{d}} -\log\left(\frac{e^{\gamma \cdot s_{y_{q,d}}}}{\sum_{y \in Y} e^{\gamma \cdot s_y}}\right)$$
where $s_{y_{q,d}}$ is the model's score for label $y_{q,d}$.
- **Problems:** These approaches do not consider the application of ranking where only the final order is important, but not the single scores. These approaches do not optimize ranking quality.

8.1.2 Pairwise

- The model scores documents independently ($f(d_i) = s_i$) but loss is based on document pairs and minimizes number of incorrect inversions; i.e. optimize regarding the total order of the documents and not specific relevance scores
- The loss can be expressed by:
$$\mathcal{L}_{ij} = \phi(s_i - s_j)$$
(more general)
- The functions could be:
  - Hinge function $\phi(z) = \max(0, 1 - z)$
  - Exponential function $\phi(z) = e^{-z}$
  - Logistic function $\phi(z) = log(1 + e^{-z})$
- **RankNet**
  - RankNet is pairwise loss function
  - predicted probabilities: $P_{ij} = P(s_i > s_j) = \dfrac{1}{1 + \exp(-\gamma(s_i - s_j))}$, desired probabilities $\bar{P}_{ij} = 1$ and $\bar{P}_{ji} = 0$

- then compute cross-entropy between $\bar{P}$ and P:

$$\mathcal{L} = \sum_{d_i \succ d_j} -\bar{P}_{ij} \log(P_{ij}) - \bar{P}_{ji} \log(P_{ji}) = \sum_{d_i \succ d_j} \log(1 + \exp(-\gamma(s_i - s_j)))$$

- There is a factorization of RankNet:
- Let $S_{ij} \in \{-1, 0, 1\}$ indicate preference between d_i and d_j
- then $\bar{P}(d_i \succ d_j) = \frac{1}{2}(1 - S_{ij})$
- and $P(d_i \succ d_j) = \frac{1}{1 + \exp(-\gamma(s_i - s_j))}$
- thus $\mathcal{L}_{ij} = \frac{1}{2}(1 - S_{ij})\gamma(s_i - s_j) + \log(1 + \exp(-\gamma(s_i - s_j)))$
- Drawbacks of RankNet:
  - RankNet is based on virtual probabilities $P(d_i \succ d_j)$ (but not a big deal)

- **Drawbacks of pairwise in general:** it is much more important to get the correct ordering at the top documents than at bottom documents but pairwise loss does not make this distinction → could prefer ranking where bottom documents are correctly ordered over ranking where top documents are correctly ordered

8.1.3 Listwise

- optimize regarding ranking metrics like DCG directly. Thus, the loss could be:

$$\mathcal{L} = -nDCG(f(\cdot, \boldsymbol{\theta}))$$

- The problem is that most ranking metrics are not differentiable. There are heuristic approaches to still optimize with respect to such metrics.
- **LambdaRank**
  - for training, not costs themselves are needed but only the gradients (gradient should be bigger for pairs of documents that have a big impact on nDCG by swapping positions)
  - Multiply actual gradients with change in nDCG by swapping rank positions of two documents:

$$\lambda_{\text{LambdaRank}} = \lambda_{\text{RankNet}} \cdot |\Delta nDCG|$$

  - works also with other metrics (e.g. delta precision)
- **ListNet and ListMLE**
  - create probabilistic model for ranking, which is diff'ble
  - sample documents from Plackett-Luce distribution (without replacement i.e. after sampling, document is removed from denominator):

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

  - ListNet: compute distribution over all possible permutations based on model score and ground-truth. Then minimize KL-divergence between both distributions (very expensive so only look at topK (mostly only top1))
  - ListMLE: compute likelihood of ideal ranking based on ground-truth

8.1.4 Problems with offline Learning to Rank:

- similar to offline evaluation in Section 2.2
- All described methods require an annotated dataset which contains either relevance labels for each document-query pair or a ranking over the whole collection.
- Creating such is time consuming and expensive

- unethical to create for privacy-sensitive settings
- Impossible to personalize for a user (everyone prefers a little bit different documents). Also, annotators and users might disagree in some points $\implies$ dataset does not fully reflect user behavior
- Can change over time
- might generally not reflect user preferences

8.2 Online Learning to Rank

- Learn from implicit user feedback
- virtually free and easy to obtain
- indicative of preferences but hard to interpret
  - Might be noisy
  - Biased
    - position bias (higher rank is more frequently clicked)
    - selection bias (only a limited set of documents is presented to the user)
    - presentation bias (results are presented differently e.g. different movie sizes in Netflix)
- Online Learning to Rank methods can learn from user interactions, **and** control the results which are displayed/presented to the user
- Thus, these methods can be more efficients as they control over what data is actually gathered
- A general online learning to rank technique is visualized in Figure 31

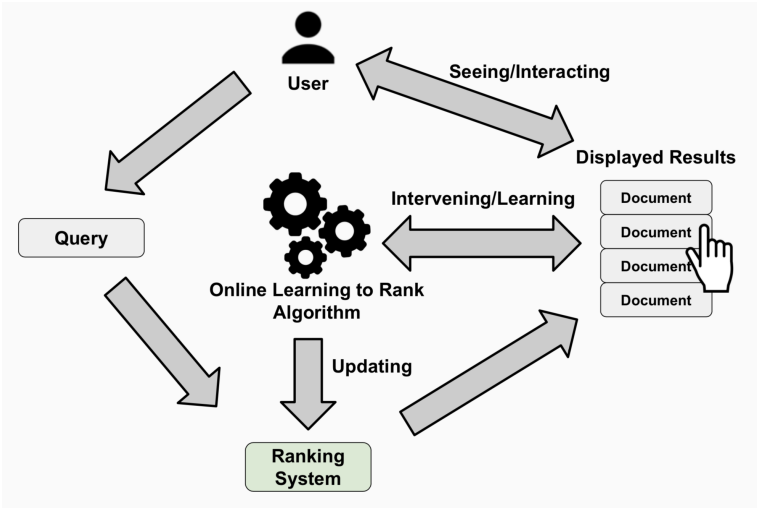


Figure 31: Overview of the general concept of online learning to rank

  - The user enters a query, for which the ranking algorithm generates a list of documents
  - The Online Learning to Rank system interacts with the results by adding and/or removing documents from the ranking. This can also include interleaving with another, slightly changed ranking algorithm
  - User interacts with the displayed result and gives implicit feedback.
  - The Online Learning to Rank algorithm updates the ranking parameters according to the analyzed feedback
- **Advantages**: learns directly from the user, is more responsive by immediately adapting its parameters
- **Risks**:
  - Unreliable methods will affect/worsen user experiences immediately.
  - (Noisy) clicks can easily bias or even manipulate search engines
  - **Self-confirming loop**

    - If an irrelevant document was clicked by random, the system still perceives that this document is relevant and will change its parameters accordingly
    - Thus, the random document will be placed higher in future ranks. However, also similar documents to the irrelevant one will have an increased relevance score and will probably occur at a high position
    - Most likely, the next clicked document will be one of the highest ones which were irrelevant $\implies$ entering a self-confirming loop
    - Due to bias and noise, an irrelevant document was clicked and inferred to be relevant
    - Due to noise, this inference is most likely to appear again
    - The algorithms confidence in this incorrect inference continues to increase

- To prevent a self-confirming loop, we have to balance exploration and exploitation
  - *Exploration*: collect feedback for learning from the most documents as possible
  - *Exploitation*: utilize what has been already learned
  - If systems only exploits, it misses out to obtain feedback for other documents that might be even better (danger to enter/staying in self-confirming loop)
  - To high exploration rate leads to a lot of irrelevant documents in ranking that worsen the user experience

8.2.1 Designing an Online Learning to Rank algorithm

- To design a OLTR algorithm, we have to make design choices in four aspects (see Figure 32)

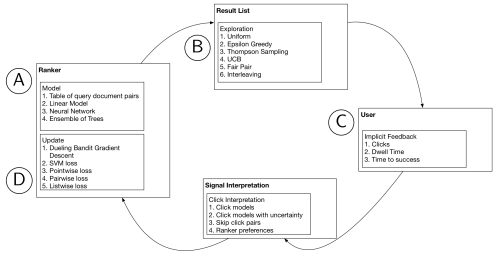


Figure 32: General design components of an OLTR algorithm

(A) **Ranker**: the ranker maps documents to relevance scores. This module operates on feature level/document id's and can be for example a linear ranker/neural model/...

(B) **Exploration strategy**: define interactions with results of the ranker. No exploration would mean that the document ranking is simply passed and stays unchanged. A common strategy is *epsilon-greedy* where we inject random documents in random positions with ratio ϵ. Other algorithms include upper confidence bound etc.

(C) **Signal recording and interpretation**: algorithm can consider multiple signals (raw observation like clicks and dwell time, more complex metrics like time to success). Should remove bias/noise. When result list was constructed by using interleaving, the feedback would also consider which ranker has won based on user interactions.

(D) **Update mechanism**: update ranking algorithm given the user feedback. If ranker operates on document id's, we can update the document's specific relevance estimate for the query. If the ranker relies on features, we optimize a loss function like the ones shown in offline LTR.

8.2.2 Dueling Bandit Gradient Descent

- One of the first OLTR algorithms was the *Dueling Bandit Gradient Descent*
- The intuition is that we compare two rankers by online evaluation, and optimize our system towards the better performing one.
- The method is structured in following steps (visualized in Figure 33)

1. From current feature state θ_b of the ranker (shown in green), sample a new ranker/feature point $\theta_c = \theta_b + u$ laying on the unit sphere $||u|| = 1$ around the current one (shown in red)
2. Get the rankings of θ_b and θ_c
3. Compare θ_b and θ_c using interleaving
4. If θ_c wins comparison: update the current model by $\theta_b \leftarrow \theta_b + \eta(\theta_c - \theta_b)$

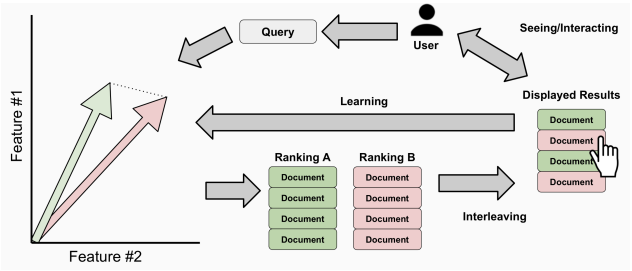


Figure 33: Steps in Dueling Bandit Gradient Descent

- It can be shown that if there is only a single optimum, the Dueling Bandit Gradient Descent algorithm will be able to approximate the optimal model

9 Counterfactual Evaluation and Learning to Rank

- The term *counterfactual* relates to *off-policy* learning in RL

9.1 Counterfactual Evaluation

- Evaluation is important before deploying a system
- *Counterfactual evaluation*: perform offline evaluation of online metrics given online data from another already deployed system
- Note that we only have partial information feedback, and no complete supervision. Only for the chosen action, we know the feedback signal/user utility. Thus, the "correct"/optimal action is unknown (also called "bandit feedback" as it was sampled from only one arm)
- Counterfactual Evaluation: Full information
  - we know true relevance labels $y(d_i)$ for all i and thus can compose additive IR metric as:

$$\Delta(f_\theta, D, y) = \sum_{d_i \in D} \lambda(\text{rank}(d_i|f_\theta, D)) \cdot y(d_i)$$

  - where λ is a weighting function, e.g.
    - average relevant position ARP: $\lambda(r) = r$
    - discounted cumulative gain DCG: $\lambda(r) = \dfrac{1}{\log_2(1+r)}$
    - precision at k: $\lambda(r) = \dfrac{\mathbf{1}[r \leq k]}{k}$
- Counterfactual Evaluation: partial Information
  - we do not know the true relevance labels $y(d_i)$ but can observe implicit biased and noisy feedback through clicks
  - Naive Estimator:
    - use click instead of relevance label

$$\Delta_{\text{Naive}}(f_\theta, D, y) = \sum_{d_i \in D} \lambda(\text{rank}(d_i|f_\theta, D)) \cdot y(d_i)$$

* even if no noise is present, the estimator $P(c_i = 1|o_i = 1, y(d_i)) = y(d_i)$ is biased by examination probabilities:

$$\begin{aligned}\mathbb{E}_o[\Delta_{\text{Naive}}(f_\theta, D, c)] &= \mathbb{E}_o\Big[\sum_{d_i \in D} c_i \cdot \lambda(rank(d_i|f_\theta, D))\Big] \\ &= \mathbb{E}_o\Big[\sum_{d_i \in D} o_i \cdot y(d_i) \cdot \lambda(rank(d_i|f_\theta, D))\Big] \\ &= \sum_{d_i \in D} P(o_i = 1|R, d_i) \cdot \lambda(rank(d_i|f_\theta, D)) \cdot y(d_i)\end{aligned}$$

* thus estimator weights documents according to examination probabilities during logging → position bias as higher rankings are more likely to be observed
* This leads to self-confirming behaviour: documents that were ranked high during logging received a lot of clicks. This leads to incorrectly assuming these documents are more relevant as they are weighted more in the evaluation

9.1.1 Inverse Propensity Scoring

- accounts for bias:

$$\Delta_{IPS}(f_\theta, D, c) = \sum_{d_i \in D} \frac{\lambda(rank(d_i|f_\theta, D))}{P(o_i = 1|R, d_i)} \cdot c_i$$

  where c_i is observed click from the log and $P(o_i = 1|R, d_i)$ is examination probability in R during logging
- this is unbiased of any additive linearly decomposable metric

9.2 Counterfactual Learning to Rank

- Rank-SVM optimizes differentiable upper bound on rank:

$$\begin{aligned}rank(d|f_\theta, D) &= \sum_{d' \in R} \mathbf{1}[f_\theta(d) \leq f_\theta(d')] \\ &\leq \sum_{d' \in R} \max(1 - (f_\theta(d) - f_\theta(d')), 0) = \overline{rank}(d|f_\theta, D)\end{aligned}$$

  Then for average relevance position metric:

$$\Delta_{ARP}(f_\theta, D, y) = \sum_{d_i \in D} rank(d_i|f_\theta, D) \cdot y(d_i)$$

  we have an unbiased estimator and differentiable upper bound

$$\begin{aligned}\Delta_{ARP-IPS}(f_\theta, D, y) &= \sum_{d_i \in D} \frac{rank(d_i|f_\theta, D)}{P(o_i = 1|R, d_i)} \cdot c_i \\ &\leq \sum_{d_i \in D} \frac{\overline{rank}(d_i|f_\theta, D)}{P(o_i = 1|R, d_i)} \cdot c_i\end{aligned}$$

- Similar approach for additive metrics: If λ is monotonically decreasing, then

$$rank(d|\cdot) \leq \overline{rank}(d|\cdot) \Rightarrow \lambda(rank(d|\cdot)) \geq \lambda(rank(d|\cdot))$$

  which provides lower bound e.g. for DCG we use:

$$\frac{1}{\log_2(1 + rank(d|\cdot))} \geq \frac{1}{\log_2(1 + \overline{rank}(d|\cdot))}$$

  such that for DCG:

$$\Delta_{DCG}(f_\theta, D, y) = \sum_{d_i \in D} \log_2(1 + rank(d_i|f_\theta, D))^{-1} \cdot y(d_i)$$

  we obtain unbiased etimator and differentiable lower bound

$$\begin{aligned}\Delta_{DCG-IPS}(f_\theta, D, y) &= \sum_{d_i \in D} \frac{\log_2(1 + rank(d_i|f_\theta, D))^{-1}}{P(o_i = 1|R, d_i)} \cdot c_i \\ &\geq \sum_{d_i \in D} \frac{\log_2(1 + \overline{rank}(d_i|f_\theta, D))^{-1}}{P(o_i = 1|R, d_i)} \cdot c_i\end{aligned}$$

- In total, propensity-weighted LTR takes the following steps:
  1. obtain model of position bias
  2. acquire large click-log
  3. for each click in log:
     (a) compute propensity of click $P(o_i = 1|R, d_i)$
     (b) calculate gradient of the bound of the unbiased estimator $\nabla_\theta \left[\frac{\lambda(\overline{rank}(d_i|f_\theta, D))}{P(o_i = 1|R, d_i)} \right]$
     (c) update model using the gradient

9.2.1 Estimating the position bias

- Assumption: observation probability (propensity) only depends on rank of document: $P(o_i = 1|i)$
- Use the RandTop-n algorithm:
  1. repeat:
     (a) randomly shuffle top n items
     (b) record clicks
  2. aggregate clicks per rank
  3. normalize to obtain propensities $p_i \propto P(o_i|i)$

  – Disadvantage: user experience suffers extremely when shuffling items randomly
- Use RandPair: Choose a pivot rank k and only swap a random other document with the document at this pivot rank
- Interventional sets/harvesting: exploit inherent "randomness" in data coming from multiple rankers (e.g. A/B tests in production logs):
  – Ranker A produces ranking: d_1, d_2, d_3, d_4
  – Ranker B produces ranking: d_3, d_2, d_1, d_4
  – if you have enough of these interactions, you see enough variations to estimate position bias (i.e. instead of swapping documents by force, use swaps that result from different systems in A/B testing)
- Jointly learning and estimating:
  – recall how probability of click can be decomposed into relevance probability times observation probability
  – if we know relevance probability, we can estimate observation probability and vice versa (as seen before in IPS)
  – It is possible to jointly learn both by iterating:
    1. learn optimal ranker given correct propensity model:
    $$P(c_i = 1|o_i = 1, y(d_i)) = \frac{P(c_i = 1 \wedge o_i = 1|y(d_i), R)}{P(o_i|R, d_i)}$$
    2. learn optimal propensity model given correct ranker
    $$P(o_i|R, d_i) = \frac{P(c_i = 1 \wedge o_i = 1|y(d_i), R)}{P(c_i = 1|o_i = 1, y(d_i))}$$
  – can also use EM-algorithm instead (this is what click models do)

10 Recommender Systems

10.1 Evaluation

- if ratings are available, could use MAE/RMSE on test set
- it is more common to treat it as a ranking problem and use common ranking metrics like precision, recall, NDCG, etc.

10.2 Challenges

- Cold start (what items to recommend to new users?)
- dynamic vs static preferences // long vs short term preferences (preferences change over time but this is more difficult to model)
- exploration vs exploitation (recommend to understand preferences? recommend based on history?)

10.3 Content-based recommenders

- main idea: recommend items to user u such that they are similar to previous items $\mathcal{I}_u$ rated highly by u
- for each item, create profile vector x_i containing features (e.g. author, title, actors; for text: set of important words); important features are picked from usual heuristics e.g. tf-idf
- build user profile as weighted sum of rated item profiles:

$$x_u = \sum_{i \in \mathcal{I}_u} r_{ui} x_i$$

  different variants: normalize weights using average rating of user or use more sophisticated aggregations
- suggest items whose feature vector x_i is most similar to profile vector x_u (e.g. cosine similarity)
- Advantages:
  - user independence - does not need information about user
  - can handle unique tastes
  - unpopular items are also recommended
  - transparency - explanations are straightforward
  - cold start for items is not an issue
- Disadvantages:
  - feature engineering / domain knowledge often needed
  - overspecialization (no serendipitous recommendations (unexpected items))
  - cold start for users

10.4 Collaborative filtering

- not only consider user u but also use ratings of other users (collaborative)
- Key idea: if user u and v have rated items similarly, and user v has rated an item, user u will rate the item similarly
- content of items is no longer needed

10.4.1 Different dimensions/approaches to CF:

- Methodology: neighborhood / model-based
  - neighborhood-base: use stored ratings directly (nearest neighbours)
  - model-based: learn predictive model and mode user-item interactions $\rightarrow$ predict new/incomplete ratings
- what is being recommended: user-based / item-based
  - user-based: use other users to infer ratings of an item for a user (neighbours = users who have similar ratings)
  - item-based: use ratings of similar items that a user has rated to predict rating for given
- type of ratings: implicit /explicit
  - explicit: like / dislike / rating (might not be available)
  - implicit: time spent, clicks (available but intention unclear)

10.4.2 Neighborhood-based recommendation

- Explicit ratings, sparse (each user represented as rating vector with dimensions = items)
- consider k-nearest neighbors of u who rated item i: $\mathcal{N}_i(u)$:

$$r_{ui} = \frac{1}{|\mathcal{N}_i(u)|} \sum_{v \in \mathcal{N}_i(u)} r_{vi}$$

- could also consider that some users are more similar to user u and use the similarity score between users w_{uv} (use cosine similarity, pearson correlation etc.):

$$r_{ui} = \frac{\sum_{v \in \mathcal{N}_i(u)} w_{uv} r_{vi}}{\sum_{v \in \mathcal{N}_i(u)} |w_{uv}|}$$

- Problems:
  - a rating of 5 might have different value for different users $\rightarrow$ normalize rating per user
  - limited coverage: users can be neighbors only if they rated common items
  - rating vectors are very sparse

Use matrix factorization to solve sparsity problem:

- project user/item vectors to dense latent space by either decomposing rating or similarity matrix
- given $|\mathcal{U}| \times |\mathcal{I}|$-matrix R of rank n, approximate it by $\hat{R} = PQ^T$ of rank $k < n$, where P is a $|\mathcal{U}| \times k$ matrix of users and Q is a $|\mathcal{I}| \times k$ matrix of items:

$$err(P, Q) = ||R - PQ^T||_F^2 = \sum_{u,i} (r_{ui} - p_u q_i^T)^2$$

- use SVD of $R = U\ \Sigma V^T$ where $U_{|\mathcal{U}| \times n}, V_{|\mathcal{I}| \times n}, \Sigma_{n \times n}$ and use low-rank approximation with parameter k such that $P = U_k \Sigma_k^{1/2}$ and $Q = V_k \Sigma_k^{1/2}$ and $r_{ui} = p_u q_i^T$

Advantages

- no feature engineering needed
- does not rely on content (which might be inaccurate)
- can handle overspecialization/diversity problems
- items recommended may not have similar content

Disadvantages

- neighborhood-based: limited coverage / sparsity
- cold start
- popularity bias: popular items are more often recommended
- cannot recommend to users with unique taste (content-based methods can!)

10.5 Deep recommenders

- scalable and can learn complex interactions
- reproducibility crisis: are often not reproducable
- typical to use implicit feedback

10.5.1 MultVAE

- VAE for collaborative filtering
- users are represented as binary vectors (interaction with item (does not) exist)
- regular VAE loss $\rightarrow$ reconstruct input vector and use isotropic gaussian as prior
- At inference: encode user history and decode latent representation $\rightarrow$ after dropping all items with which the user already interacted, select item with highest output score
- Advantages
  - All advantages that come from DL: powerful flexible, scalable etc.
  - cross-pollination, adoption of advances from ML/DL
- Disadvantages
  - cold start
  - no consensus if performance gains are significant
  - reproducability
  - deep models might not be needed

10.6 Other Approaches

10.6.1 Sequential Recommendation

- interests can change over time, so look into short-term vs long-term
- could predict next item and also next basket (in online shopping)

10.6.2 Exploration vs Exploitation

- exploitation: recomment only items that are likely to interest user (what about new user?)
- exploration: recommend other items (learn user's preferences over un-encountered items; could harm user experience)

10.6.3 Conversational Recommendation

- tackles cold start, dynamic preferences
- if unsure about attribute/item, ask!