Summary Information Retrieval 1

Phillip Lippe

January 25, 2019

Contents

1 Offline evaluation

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

1.1 Collection-based evaluation

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

1.1.1 Traditional Evaluation measures

- 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 ($P@k/R@k$)
- 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 1). While the cut-off rank heads to infinity, recall goes to 1 and precision to 0. We interpolate the curves by taking the maximum value of all future values/values right from a given point.

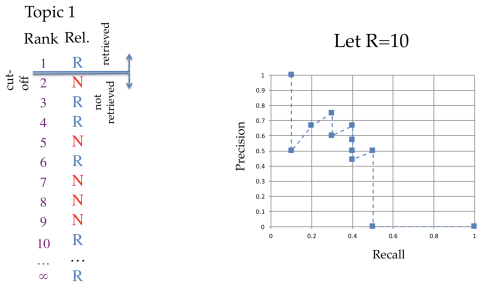


Figure 1: R/P curves for ranking

- 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
  - 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

1.1.2 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: $\theta_r = \mathcal{R}(\text{rel}_r) = \frac{2^{\text{rel}_r}-1}{2^{\max \text{rel}}}$. The corresponding utility accumulation is the *Expected Reciprocal Relevance*: $ERR@k = \sum_{r=1}^{k} \frac{1}{r} \cdot \theta_r \cdot \prod_{i=1}^{r-1} (1-\theta_i)$

1.1.3 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.

1.2 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, ...)?

1.3 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)

1.3.1 Student's t-test

- Statistic:

$$t = \frac{\mu_{E-P}}{\frac{\sigma_{E-P}}{\sqrt{N}}} = \frac{\overline{AP_E - AP_P}}{\frac{\sigma_{E-P}}{\sqrt{N}}}$$

- We assume that the mean measure follow a normal distribution (see Figure 2)

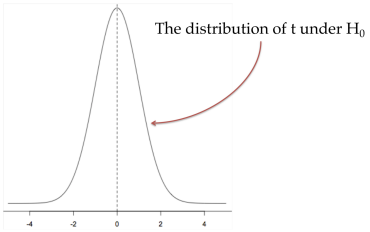


Figure 2: Distribution of t values under the null hypothesis

- The p-value is determined by the area under the distribution right from the determined value
- If the p-value is lower than the significance level α, reject null hypothesis
- There are two different error types for the t-test:

  Type 1 rejecting null hypothesis although it was true (prob. is α)

  Type 2 not rejecting the null hypothesis although it was false (prob. is β)

- There are four aspects of the test that interact with each other. If one is unknown, it can be derived from the others

  1. *Sample size* N
  2. *Effect size* = diff. of means / std. dev.
  3. *Significance level* = Type 1 error α
  4. *Power* = 1 - Type 2 error β. Prob. of finding an effect if it is there.

1.3.2 Sign test

- Look at score/sample pairs from A and B and consider the null hypothesis $H_0 : P(B > A) = P(A > B) = 1/2$.
- It is a discrete way of looking at the t-test. For a sample size of N, we get a binomial distribution with N bins. The bins summed up from the measured number of B winning over A describe the p-value (see Figure 3)

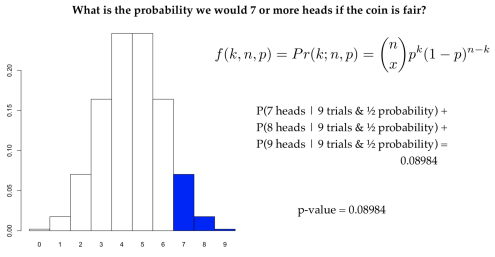


Figure 3: Distribution of t values under the null hypothesis

1.3.3 Distribution-free tests

- Tests where we do not explicitly assume the underlying data to be sampled from a specific distribution
- **Randomization test**
  - Given: a set of results for N queries for algorithm A and B

  - Repeat for many times:
    * Randomly swap values for a query in algorithm A and B
    * Compute average of both systems and their difference
    * Add difference to an array
  - The two systems are significantly different, if the actual difference without swapping is outside 95% of the differences in the array.

- **Bootstrap test**
  - Same preparation as for randomization test
  - Repeat for many times:
    * Randomly sample pair of scores (i.e. selecting queries) of A and B with replacement
    * Compute average of each systems in the set of pairs
    * Add difference to an array
  - The two systems are significantly different if the mean of the array can be shown to be significantly different from 0.

2 Online evaluation

- In online evaluation, the system interacts with the user $\implies$ user "tells" what is relevant, system analyzes the user's behavior for gaining that knowledge
- The benefit of online evaluations is that they are mostly simpler and directly incorporate measuring the ranking quality
- 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.

2.1 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.

2.2 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.
- *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?

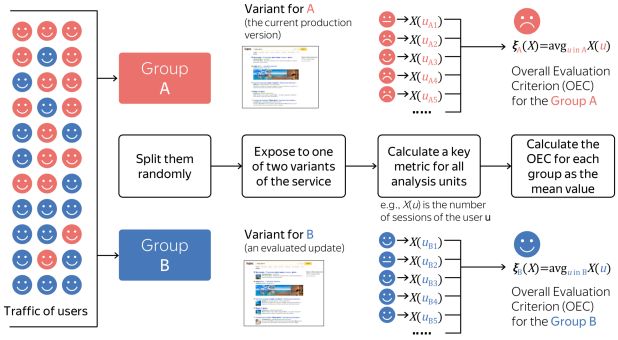


Figure 4: Visualization of A/B Testing

2.3 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

2.3.1 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
- 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

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 5: Formal algorithm describing balanced interleaving

2.3.2 Team-draft interleaving

- In team-draft interleaving we guarantee that both algorithms contribute equally to the interleaved ranking
- At each stage, we flip a coin to determine whether to pick the next document from A or B first. Afterwards, the document of the other system is picked
- If a document is already in the interleaved ranking, we look for the next document from the *same* ranking model until we find a new document.

Algorithm 2 Team Draft Interleaving

Input: Rankings $A = (a_1, a_2, ...)$ and $B = (b_1, b_2, ...)$
Init: $I = ()$; TeamA $\leftarrow \emptyset$;TeamB $\leftarrow \emptyset$
while$(\exists i: A[i] \notin I) \wedge (\exists j: B[j] \notin I)$ **do**
  if $(|TeamA| < |TeamB|) \vee$
    $((|TeamA| = |TeamB|) \wedge (RandBit() = 1))$ **then**
    $k \leftarrow min_i\{i: A[i] \notin I\}$ *...top result in A not yet in I*
    $I \leftarrow I + A[k]$ *... append it to I*
    $TeamA \leftarrow TeamA \cup \{A[k]\}$ *... clicks credited to A*
  else
    $k \leftarrow min_i\{i: B[i] \notin I\}$ *...top result in B not yet in I*
    $I \leftarrow I + B[k]$ *... append it to I*
    $TeamB \leftarrow TeamB \cup \{B[k]\}$ *... clicks credited to B*
  end if
end while
Output: Interleaved ranking *I, TeamA, TeamB*

Figure 6: Formal algorithm describing Team-draft interleaving

- There are also corner cases that can cause troubles in team-draft interleaving. However, in practice this rarely happens/has a significant effect.

2.3.3 Probabilistic interleaving

- To avoid biases completely, we can apply probabilistic models
- Convert the ranking of each model to a probability distribution by applying softmax ($\tau = 3$):

$$p_i(d) = \frac{\frac{1}{r_i(d)^\tau}}{\sum_{d' \in D} \frac{1}{r_i(d')^\tau}}$$

- For every position in the interleaved ranking, flip a coin to determine whether to pick a document from A or B. Next, we sample from the corresponding softmax distribution a document without replacement, and add it to the interleaved list. The picked document is removed from the probability distributions of A and B.
- We can perform evaluation by counting the clicks for documents sampled from A and B. We expect the same number of clicks for documents at the same position of both algorithms due to the same probability in the softmax.
- Note that compared to a hard assignment of 0 or 1 in balanced and team draft interleaving, the distribution of 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)

ALGORITHM 4: Probabilistic Interleave.

Input: l_1, l_2, τ
$\mathbf{l} \leftarrow []$
$\mathbf{a} \leftarrow []$
for $i \in (1, 2)$ **do**
  initialize $s(l_i)$ using Eq. 3
while $(\exists r : l_1[r] \notin \mathbf{l}) \vee (\exists r : l_2[r] \notin \mathbf{l})$ **do**
  $a \leftarrow 1$ **if** $random\_bit()$ **else** 2
  $\bar{a} \leftarrow 2$ **if** $a = 1$ **else** 1
  $append(\mathbf{a}, a)$
  $d_{next} \leftarrow sample\_without\_replacement(s(l_a))$
  $append(\mathbf{l}, d_{next})$
  $remove\_and\_renormalize(s(l_{\bar{a}}), d_{next})$
// *present* $\mathbf{l}$ *to user and observe clicks* $\mathbf{c}$
compute o, e.g., using Eqs. 6–9
return o

Figure 7: Formal algorithm describing probabilistic interleaving

- The algorithm is summarized in Figure 7
- Another, more efficient evaluation method is by marginalizing over all possible assignments a. Therefore, we calculate the probability of the interleaved list l given a (and the query q) by successively multiplying the softmax probabilities at that point. For example, the first assignment $a = \{1, 1, 1, 1\}$ leads to the following calculation:

$$p(l_i|a = \{1, 1, 1, 1\}, q) = 0.85 \cdot \frac{0.1}{0.15} \cdot \frac{0.03}{0.05} \cdot \frac{0.02}{0.02} = 0.34$$

- Normalizing all $p(l_i|a, q)$ by its sum lead to $p(a|l_i, q) \implies p(a|l_i, q) = \frac{p(l_i|a,q)}{\sum_{a \in A} p(l_i|a,q)}$
- For every assignment, we add the value $o = -1$ times the probability $p(a|l_i, q)$ if more clicked documents were assigned to A. If B has more clicks, we use $o = 1$ as factor, and ignore it for a tie (or multiply by $o = 0$). Thus, our expected number of wins B has more than A is given by:

$$E[O] = \sum_{a \in A} o_a \cdot p(a|l_i, q) \quad \text{where} \quad o_a = \begin{cases} -1 & \text{if } c_A > c_B \\ 0 & \text{if } c_A = c_B \\ 1 & \text{if } c_A < c_B \end{cases}$$

- Figure 8 visualizes an example for probabilistic interleaving

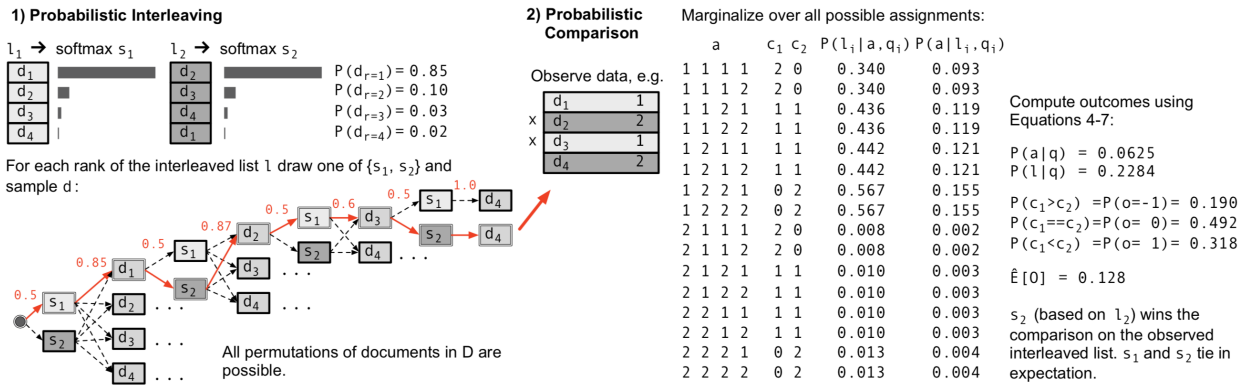


Figure 8: Visualization of probabilistic interleaving

3 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,...) ⇒ 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

3.1 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: $P(C_u = 1) = \rho_{uq}$

3.2 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 9

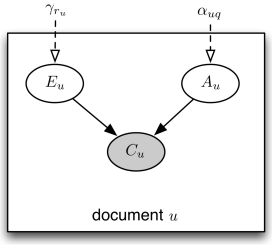


Figure 9: Probabilistic graphical model of parameters for PBM

- 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

3.3 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 10

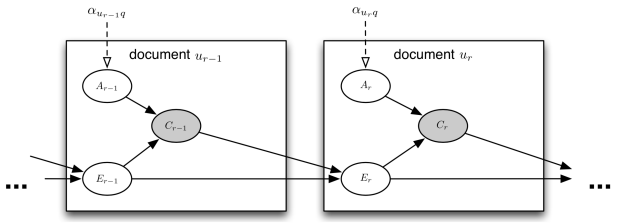


Figure 10: Probabilistic graphical model of parameters for CM

- *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

4 Introduction to Retrieval models

- Mathematical framework for defining query-document matching

4.1 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 (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.

4.2 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

4.3 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 by its similarity. Therefore we can use either document likelihood, query likelihood or KL-divergence

4.3.1 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 $p(d|q) \propto p(q|d)$
- 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. There are mainly three modeling choices:

1. *How to define the generative process $p|\theta_d$?*
   - Given θ_d, what is the generative process for getting $q = w_1, ..., w_{|q|}$?
   - Different distributions are possible
   - *Multiple Bernoulli* - bag of word perspective, every word in vocabulary has probability to be in query or not. The related probability is:

$$p(q|\theta_d) = \prod_{w_i \in q} p(X_i = 1|\theta_d) \prod_{w_i \notin q} \left(1 - p\left(X_i = 1|\theta_d\right)\right)$$

   - *Multinomial* - similar to bernoulli, but we know have a random variable for every word slot in the query and not one for every word in the vocabulary. Thus, the calculation is:

$$p(q|\theta_d) = \prod_{w_i \in q} p(w_i|\theta_d) \quad \text{where} \quad \sum_{w_i \in V} p(w_i|\theta_d) = 1$$

   - *Multiple Poisson* - similar to bernoulli, but instead of presence or absence, we model the number of times we expect a word from the vocabulary to occur in the query of length $|q|$ by a Poisson distribution:

$$p(q|\theta_d) = \prod_{w_i \in V} \frac{e^{-\lambda_i |q|} (\lambda_i |q|)^{\text{tf}(w_i;d)}}{\text{tf}(w_i; d)!}$$

2. *How to estimate θ_d based on document d?*
   - To estimate θ_d we perform MLE: $\hat{\theta}_d = \arg\max_{\theta_d} p(d|\theta_d)$
   - In case of a multinomial distribution, we would get:

$$p(d|\theta_d) = \prod_{w_i \in V} p(w_i|\theta_d)^{\text{tf}(w_i;d)} \implies \log p(d|\theta_d) = \sum_{w_i \in V} \text{tf}(w_i; d) \log p(w_i|\theta_d)$$

   - Note that this is a constrained optimization problem with $\sum_{w_i \in V} p(w_i|\theta_d) = 1$.
   - By using lagrangian multiplier, we get $p_{MLE}(w_i|d) = \frac{\text{tf}(w_i;d)}{|d|}$
3. *How to compute prior $p(d)$?*
   - The prior takes everything into account which is independent of a query.
   - This can include number of clicks, credibility, ...

4.3.2 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 $p(w_i|\theta_d) = \frac{\text{tf}(w_i;d)}{|d|}$
- **Adaptive smoothing**: add a small extra count to every word:

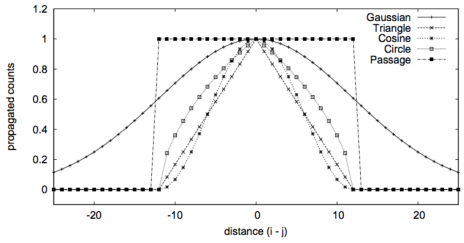
$$p(w_i|\theta_d) = \frac{\text{tf}(w_i; d) + \epsilon}{|d| + \epsilon|V|}$$

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:
$$p_\lambda(w_i|\theta_d) = \lambda \frac{\text{tf}(w_i; d)}{|d|} + (1-\lambda)\frac{\text{tf}(w_i; C)}{|C|}$$
The context C is approximated by the concatenation of all documents.
- **Dirichlet prior smoothing**: we assume that before seeing the document, we have a prior belief over all words $p(\theta_d)$. 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!):
$$p(w|\theta_d) = \frac{\text{tf}(w; d) + \alpha_w - 1}{\alpha_w + \beta_w - 1}$$

4.3.3 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



5 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

5.1 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 rows 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.

$\mathbf{X}_{n\times m}$

	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

=

ship	−0.44	−0.30	0.57	0.58	0.25
boat	−0.13	−0.33	−0.59	0.00	0.73
ocean	−0.48	−0.51	−0.37	0.00	−0.61
wood	−0.70	0.35	0.15	−0.58	0.16
tree	−0.26	0.65	−0.41	0.58	−0.09

$U_{n\times r}$

2.16	0.00	0.00	0.00	0.00
0.00	1.59	0.00	0.00	0.00
0.00	0.00	1.28	0.00	0.00
0.00	0.00	0.00	1.00	0.00
0.00	0.00	0.00	0.00	0.39

$\Sigma_{r\times r}$

$V_{m\times r}{}^T$

d_1	d_2	d_3	d_4	d_5	d_6
−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
−0.29	−0.53	−0.19	0.63	0.22	0.41
0.28	−0.75	0.45	−0.20	0.12	−0.33
0.00	0.00	0.58	0.00	−0.58	0.58
−0.53	0.29	0.63	0.19	0.41	−0.22

Figure 11: Example of SVD for 6 documents and 5 terms

- 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.

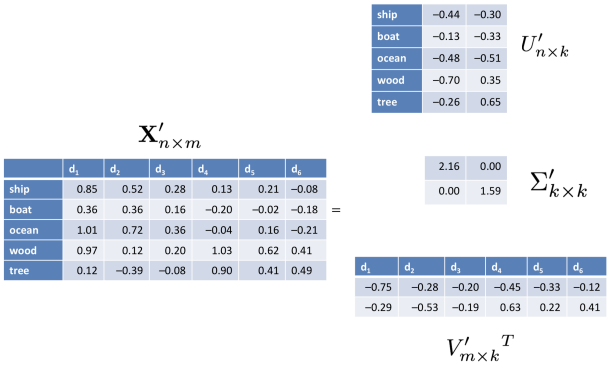


Figure 12: Reduced dimensions by $k = 2$ for 6 documents and 5 terms

- In case of Figure 11 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 12.
- 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)

5.2 Probabilistic Latent Semantic Indexing

- (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 13.

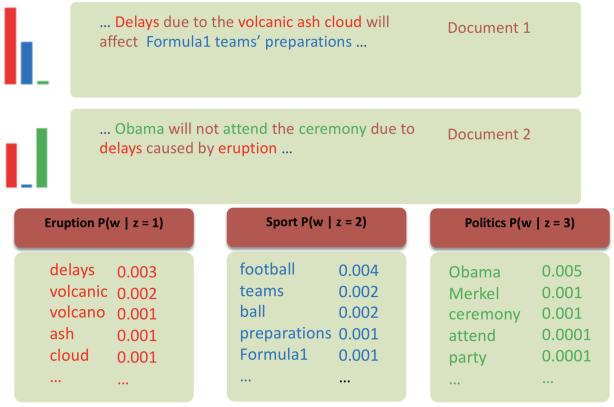


Figure 13: Example of document representation in PLSI

- For semantic matching, we calculate the cosine similarity between the topic distributions of the query and the document.
- PSLA 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

5.2.1 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

5.2.2 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.

5.2.3 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 14 visualizes the graphical model of the Latent Dirichlet Allocation

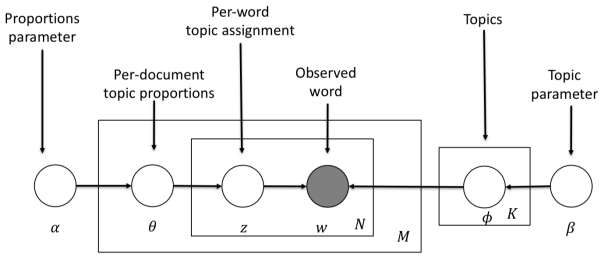


Figure 14: Graphical model of LDA

- In this diagrams, it is easier to show extensions. For instance, Figure 15 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

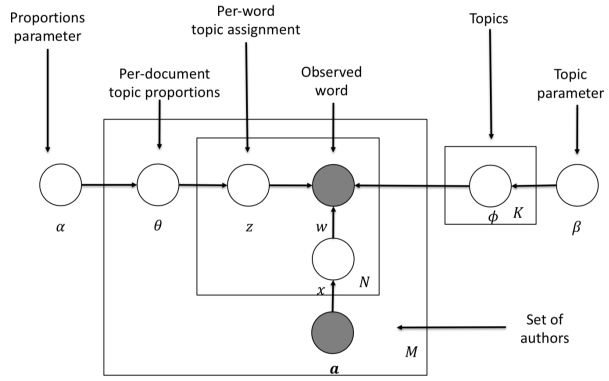


Figure 15: Graphical model of the Author-topic model

5.3 Probabilistic Topic Models in IR

1. **Topic matching**
   - Represent document by $p(z|d)$ and query by $p(z|q)$
   - Take the KL divergence or similar measure to find score/similarity between the distributions
   - But: query is very short so that topic model is harder to infer without having too much noise
2. **Smoothing**
   - Smooth probabilities according to the topics in the document:

$$\begin{aligned} p(w|d) &= \lambda p_\mu(w|d) + (1-\lambda) p_{\text{tm}}(w|d) \\ &= \lambda p_\mu(w|d) + (1-\lambda) \left(\sum_{z=1}^{K} p(w|z) p(z|d) \right) \end{aligned}$$

   - Thus we apply Jelinek-Mercer smoothing where the context is replaced by the topic word distributions of the document
3. **Query expansion**
   - We can build an own language model for a given query by using the word distributions of the topics:

$$p_{\text{tm}}(w|q) = \sum_{z=1}^{K} p(w|z) p(z|q)$$

6 Neural Retrieval Models

6.1 Distributed Word Representations

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

6.1.1 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 16)

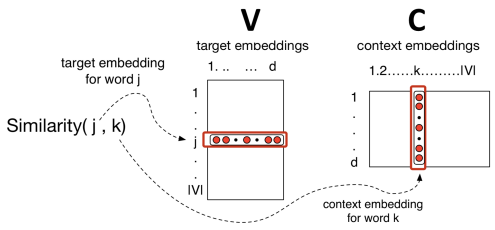


Figure 16: Visualization of skip gram method for learning word representations

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

6.1.2 Using word embeddings in IR

- **Generalized Language Model**
  - The standard language model assume that a term t_q occurring in q is being sampled from a document or a background collection (smoothing):

$$p_{LM}(t_q|d) = \lambda \cdot p(t_q|d) + (1-\lambda) \cdot p(t_q|C)$$

  - The generalized language model extends this idea by also considering terms that are similar to t_q (for example synonyms):

$$p_{LM}(t_q|d) = \lambda \cdot p(t_q|d) + \alpha \sum_{t' \in d} p(t_q|t', d)p(t'|d) + \beta \sum_{t' \in N_t} p(t_q|t', C)p(t'|C) + (1-\alpha-\beta-\lambda) \cdot p(t_q|C)$$

    where

$$p(t_q|t', d) = \frac{sim(t', t_q)}{\sum_{t'' \in d} sim(t', t'')} \text{ and } p(t'|d) = \frac{tf(t'; d)}{|d|}$$

    N_t is the set of the most similar words to t_q.

- **Word Mover's distance**
  - For every word w_i in the query q, look for the word with the highest similarity/smallest distance in document d
  - Score a document by the sum of the pairwise distances. The document with the smallest distance gets the highest rank
  - However, this approach doesn't care about the whole document but only the best matches

6.2 Compositionality

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

6.2.1 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

6.2.2 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 17
  - Second method: *Distributed bag of words*

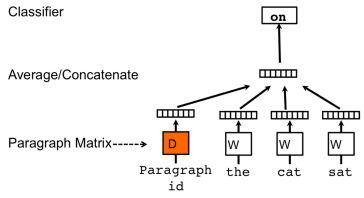


Figure 17: Distributed memory model.

* 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 18

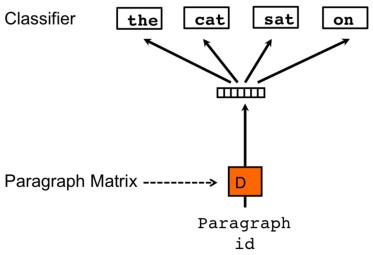


Figure 18: Distributed BOW model.

- **Lexicographical definition**
  - We can also use the lexicographical definitions of words to train and/or test the word embeddings
  - The word embeddings of the definition are combined by an (arithmetic) function f_c, and compared to the embedding of the word to be defined
  - The objective is to minimize the distance to the defined word, but maximize the distance to other words to distinguish between words
  - An example is shown in Figure 19

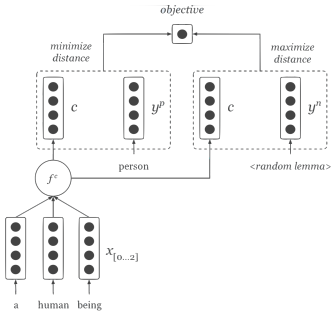


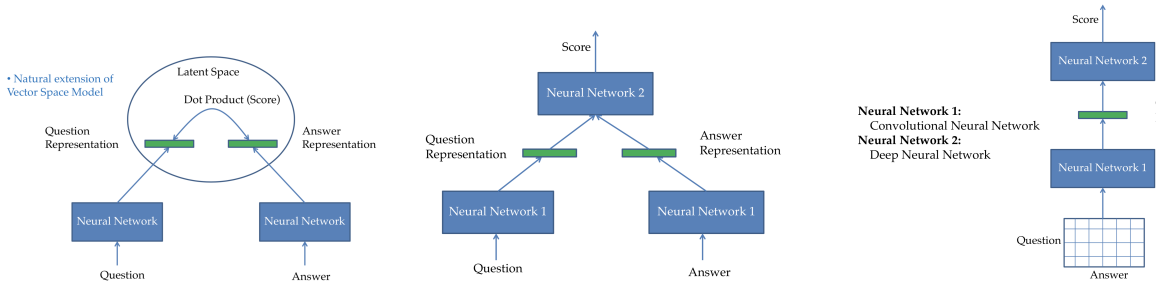
Figure 19: Lexicographical model for the example of "*person*" defined as "*a human being*".

6.2.3 Tune word vectors and learn rules of composition

- Deep, neural architectures which have different aspects to be designed
- **Architectures and representation**
  - The simplest approach is to use neural networks to embed documents and queries to latent space, and then perform same similarity measures as before (like cosine similarity). This method is also referred to as *Projection to latent space* (see Figure 20a). Possible network architectures are Convolutional NN, Recurrent NN or fixed Deep NNs if (max) input size is known

– The next step is to replace the similarity measure by another neural network. Thus, this NN takes the composed embeddings of the document and query as input, and return a single real value indicating the similarity score. This approach is called *One Dimensional Matching* and visualized in Figure 20b. The common architecture for the highest

– Another architecture is spanning up a two-dimensional input by the query and document. Therefore, we compute similarity scores for every word in the query to every word in the document which results in a two dimensional matrix. On this, we apply a convolutional NN to end up in a fixed-size embedding. A consecutive fully-connected NN maps this embedding into a similarity score. Figure 20c visualizes



(a) Projection to Latent space (b) One dimensional matching (c) Two dimensional matching

Figure 20: Comparison of different neural architectures for compositionality

- **Training**: depending on the available data, we can perform different levels of supervision
  - *No supervision/labels*: If no labels are provided at all, we could autoencoders to reduce query and document to a latent space and check similarity metrics. Otherwise, we can make use of pretrained neural language models with techniques like ELMo and BERT
  - *Distant supervision*: We create pseudo-labels by sampling short word sequences from a document and considering this as query. The document from which we sampled is labeled as relevant/high similarity, while all other documents (from which we sample one randomly for training) are considered as being not relevant
  - *Weak supervision*: As alternative, we can use unsupervised ranking functions like BM25 to generate labels and use this scores for training (teacher-student architecture). In experiments, the neural network was even able to outperform BM25.
  - *Full supervision*: Labels are created by either annotators or using the click log as implicit feedback from the users. We can train the models in a standard supervised fashion.

7 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 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)

7.1 Offline Learning To Rank

- Given an annotated dataset of relation document and relevance/ranking
- There are three different approaches

1. **Pointwise**: optimize models $f(\boldsymbol{d}, \boldsymbol{\theta})$ to predict relevancy of a document. This can be recasted in a regression problem with loss:

$$\mathcal{L} = \sum_{\boldsymbol{d}} \left(f(\boldsymbol{d}, \boldsymbol{\theta}) - \text{relevancy}(d, q)\right)^2$$

   However, this approach does not consider the application of ranking where only the final order is important, but not the single scores.

2. **Pairwise**: optimize regarding the total order of the documents and not specific relevance scores. The loss can be expressed by:

$$\mathcal{L} = \sum_{d \succ d'} \left[f(\boldsymbol{d'}, \boldsymbol{\theta}) - f(\boldsymbol{d}, \boldsymbol{\theta})\right]$$

   where $d \succ d'$ means that d' is the successor of d in the labeled ranking. Nevertheless, this method does not take into account that only a subpart (top 10) of the collection is actually presented to the user.

3. **Listwise**: optimize regarding ranking metrics like DCG. 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.

- Problems with offline Learning to Rank: similar to offline evaluation in Section 1.2
  - 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
  - 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

7.2 Online Learning to Rank

- Learn from implicit user feedback
  - Might be noisy
  - Consider position bias (higher rank is more frequently clicked) and selection bias (only a limited set of documents is presented to the user)
- 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 21

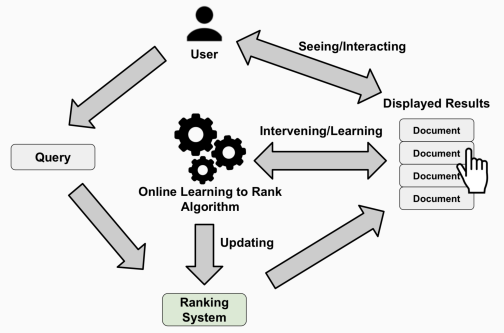


Figure 21: Overview of the general concept of online learning to rank

  - 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 $\Longrightarrow$ 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

7.2.1 Designing an Online Learning to Rank algorithm

- To design a OLTR algorithm, we have to make design choices in four aspects (see Figure 22)

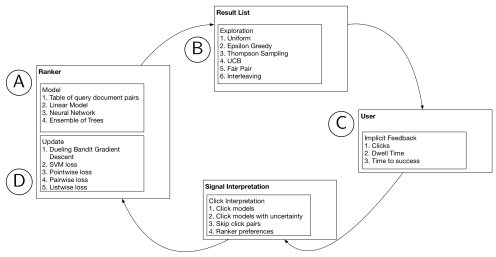


Figure 22: General design components of an OLTR algorithm

(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.

7.2.2 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 23)
  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 $\theta_b \leftarrow \theta_b + \eta(\theta_c - \theta_b)$

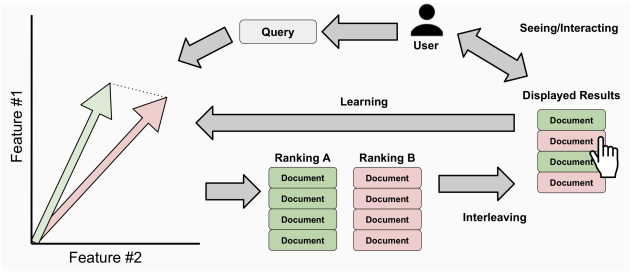


Figure 23: Steps in Dueling Bandit Gradient Descent

- 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

8 Counterfactual Evaluation and Learning to Rank

- The term *counterfactual* relates to *off-policy* learning in RL
- Thus, we try to evaluate an offline task by using online data obtained by another policy to estimate the performance of the new policy in a online setting

8.1 Counterfactual Evaluation

- In general, a user interactive system can be formalized as follows (see Figure 24):
  - x: Feature vector describing the user and context (i.e. query)
  - y: Result the system returns based on its policy ($y = \pi(x)$)
  - δ: Feedback signal from the actions a user took. The function encodes the metric (user utility function) and is defined as $\delta : X \times Y \rightarrow \mathbb{R}$
  - π: Policy describing the ranking system which takes x as input and maps it to output y: $\pi : X \rightarrow Y$
- *Counterfactual evaluation*: perform offline evaluation of online metrics given online data from another system $\pi_{\text{production}}$. Thus, we try to estimate performance of π_{new} with interaction data obtained with $\pi_{\text{production}}$.
- The interactions data/log is structured as $D = \{(x_1, y_1, \delta_1), ..., (x_n, y_n, \delta_n)\}$
  - The actions y_i were selected by $\pi_{\text{production}} : X \rightarrow Y$
  - 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)
- We want to estimate $\mathbb{E}_{y \sim \pi_{\text{new}}}[\delta(x, y)]$ given D from $\pi_{\text{production}}$. For this, there are two approaches: *model the rewards* and *inverse propensity scoring*.

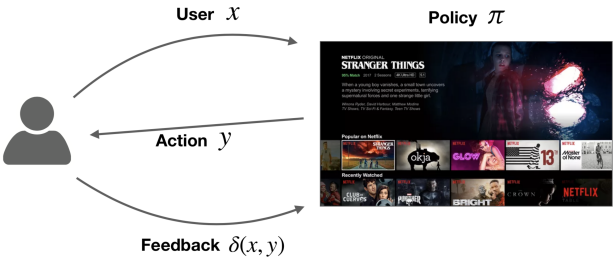


Figure 24: Visualization fo a user interactive system

8.1.1 Model the rewards

- The intuition behind *model the rewards* is to learn the reward function $\delta : X \times Y \to \mathbb{R}$ from $D \sim \pi_{\text{production}}$ directly
- The task can be reduced to a regression problem:

$$\delta_w = \arg\min_{\delta_w} \sum_{i=1}^{N} \mathcal{L}\left(\delta_w\left(x_i, y_i\right), \delta_i\right)$$

  where $\mathcal{L}$ is a loss function like MSE.
- Once δ_w is learned, we can estimate our goal by $\mathbb{E}_{y\sim\pi_{\text{new}}}\left[\delta(x,y)\right] = \frac{1}{n}\sum_{i=1}^{N} \delta_w\left(x_i, \pi_{\text{new}}(x_i)\right)$
- However, learning δ_w is in general very difficult, as:
  - Input space $X \times Y$ is very high-dimensional
  - Rewards are highly non-linear and noisy
  - Data is strongly biased to the actions that $\pi_{\text{production}}$ prefers

8.1.2 Inverse Propensity Scoring

- Instead of learning δ_w, is it possible to directly estimate the value of the new policy π_{new}?
- Answer: only under the condition that the policy $\pi_{\text{production}}$ is stochastic: $y \sim \pi(y|x)$. The probability p to choose the action y is also called *propensity*. Note that $p > 0$ must hold for all possible actions as we otherwise have no chance to discover/obtain feedback for all actions
- For unbiased counterfactual evaluation, we need data samples with the propensity p_i from policy $\pi_{\text{production}}$ describing the probability of selecting y_i for given input x_i: $(x_i, y_i, \delta_i, p_i)$
- Use importance sampling to make distributions $\pi_{\text{production}}$ and π_{new} comparable. This leads to the **IPS-estimator**:

$$\frac{1}{n}\sum_{i=1}^{N} \delta_i \frac{\pi_{\text{new}}(y_i|x_i)}{p_i}$$

8.1.3 Proof of Unbiasedness

- We want to proof that in expectation, the IPS estimator will lead to the correct value:

$$\mathbb{E}_{y\sim\pi_{\text{production}}}\left[\frac{1}{n}\sum_{i=1}^{N} \delta_i \frac{\pi_{\text{new}}(y_i|x_i)}{p_i}\right] = \mathbb{E}_{y\sim\pi_{\text{new}}}\left[\delta(x,y)\right]$$

- First, we can put the sum outside the expectation:

$$\mathbb{E}_{y\sim\pi_{\text{production}}}\left[\frac{1}{n}\sum_{i=1}^{N} \delta_i \frac{\pi_{\text{new}}(y_i|x_i)}{p_i}\right] = \frac{1}{n}\sum_{i=1}^{N} \mathbb{E}_{y\sim\pi_{\text{production}}}\left[\delta_i \frac{\pi_{\text{new}}(y_i|x_i)}{p_i}\right]$$

- Next, we replace the expectation by a sum over actions weighted by their corresponding probabilities:

$$\frac{1}{n}\sum_{i=1}^{N}\mathbb{E}_{y\sim\pi_{\text{production}}}\left[\delta_i\frac{\pi_{\text{new}}(y_i|x_i)}{p_i}\right]=\frac{1}{n}\sum_{i=1}^{N}\sum_{y_i\in Y}\left[\pi_{\text{production}}(y_i|x_i)\delta_i\frac{\pi_{\text{new}}(y_i|x_i)}{p_i}\right]$$

- As p_i is defined as $\pi_{\text{production}}(y_i|x_i)$, we can reduce the equation to:

$$\frac{1}{n}\sum_{i=1}^{N}\sum_{y_i\in Y}\left[\pi_{\text{production}}(y_i|x_i)\delta_i\frac{\pi_{\text{new}}(y_i|x_i)}{p_i}\right]=\frac{1}{n}\sum_{i=1}^{N}\sum_{y_i\in Y}\left[\delta_i\pi_{\text{new}}(y_i|x_i)\right]$$

- Finally, we apply rules based on the definition of expectation:

$$\frac{1}{n}\sum_{i=1}^{N}\sum_{y_i\in Y}\left[\delta_i\pi_{\text{new}}(y_i|x_i)\right]=\frac{1}{n}\sum_{i=1}^{N}\mathbb{E}_{y_i\sim\pi_{\text{new}}}\left[\delta(x_i,y_i)\right]=\mathbb{E}_{y\sim\pi_{\text{new}}}\left[\delta(x,y)\right]$$

- Note that the IPS estimator has a high variance which scales with p_i^2. Thus, if we have a very low probability for some actions, this can introduce a high error $\implies$ many samples needed to approximate target accurately. There are different approaches to reduce the variance

8.2 Counterfactual Learning to Rank

- Learning to Rank: *offline* - train on labeled data, *online* - learn from user interactions, *counterfactual* - learn offline from online retrieved data obtained by another policy/ranker
- The goal of counterfactual LTR is to learn a new ranker π_{new} from the interaction data with $\pi_{\text{production}}$
  - The data is specified by $D=\{(x_1,y_1,\delta_1),...,(x_N,y_N,\delta_N)\}$ where δ_i indicates which document was clicked (we assume that only one document was clicked)
  - y_i is the ranking selected by $\pi_{\text{production}}: X\to Y$
- Naive approach: assume click indicates relevance and learn as if it would be a supervised dataset:

$$\pi_{\text{new}}=\arg\min_{\pi}\sum_{i=1}^{N}\text{rank}\left(\pi(x_i),y_i,\delta_i\right)$$

  The objective function is to reduce the rank of the relevant document given the new ranking of π_{new} and the previous ranking by y_i. Can be solved by pairwise LTR objective.
- However, data obtained by online Learning to Rank is commonly noisy and biased
- We can take these biases into account by using the inverse propensity scores:

$$\pi_{\text{new}}=\arg\min_{\pi}\sum_{i=1}^{N}\frac{\text{rank}\left(\pi(x_i),y_i,\delta_i\right)}{p(\textit{observing }\delta_i)}$$

  This formula can be motivated from a probabilistic click model perspective:

$$p(\textit{click})=p(\textit{observation})\times p(\textit{relevant})\implies p(\textit{relevant})=\frac{p(\textit{click})}{p(\textit{observation})}$$

  Left side is what we want to get, and on the right side it is specified what we actually optimize.

8.2.1 Propensity estimation

- However, the question remains how we calculate $p(\textit{observing }\delta_i)$. We can either approximate it by using click models, or by performing a randomization test
- *RandTopN*
  - Randomly shuffle the top N documents
  - Measure clicks people have performed on the data (online experiment)
  - Aggregate clicks for infinite samples

- Infer $\hat{p} \propto p(\textit{observing}\delta_i)$

- *RandPair*
  - Randomly swap top document with random top N documents
  - Infers $\frac{p(\textit{observing}\delta_i)}{p(\textit{observing}\delta_j)}$ for swapped documents i and j

(a) RandTopN (b) RandPair

8.2.2 The Variance problem

- The problem of solving the counterfactual approach is that if $p(\textit{observing}\delta_i)$ heads to 0, the overall objective will be heavily biased towards this example $\implies$ overfitting on single data point
- One way to overcome this problem is using a variance regularizer which prevents the policy to deviate too much from the original production policy $\pi_{\text{production}}$:

$$\pi_{\text{new}} = \arg\min_{\pi} \sum_{i=1}^{N} \frac{\text{rank}\,(\pi(x_i), y_i, \delta_i)}{p(\textit{observing}\ \delta_i)} + \lambda \sqrt{\frac{\mathcal{V}[\pi, \pi_{\text{production}}]}{n}}$$

- However, this optimization problem cannot be solved by SGD anymore and iterative methods must be applied (new learning framework *counterfactual risk minimization*)