universität freiburg

Databases and Information Systems WS 25/26

Lecture 2: Ranking and Evaluation

October 21, 2025

Prof. Dr. Hannah Bast
Department of Computer Science
University of Freiburg

Overview of this lecture

- Organizational
  - Your experiences with ES1 — Inverted index
  - Your tutor — Feedback and timeslots
- Contents
  - Ranking — tf.idf and BM25, other tricks
  - Evaluation — Ground truth, Precision, AP, MAP, nDCG, BPref, Overfitting
  - **ES2:** Implement BM25, tune your search engine using a training benchmark, then evaluate on a test benchmark

    There will be a small competition for your enjoyment and motivation (in the form of a table on the Wiki)

Experiences with ES1 1/2

- Excerpts from your feedback [somewhat representative]

  "I really liked the lecture and the exercise sheet"

  "Nice, gentle introduction back into study mode after the break"

  "The first lecture was easy for me, nothing really complicated"

  "Ich finde es toll, dass die Übungen optional sind"

  "I like the lecture and especially that it is uploaded on YouTube"

  "I only want the points and a Haiku rating my performance"

  "The coding part was a bit boring to follow"

  "War für mich recht schwierig wieder reinzukommen"

  "Sometimes I wished the first exercise sheet was more difficult so that I would not start every semester illusionized and only realized at the end that I am stupid or whatever"

Experiences with ES1 2/2

- Queries from your feedback
  - Queries that worked well:

oppenheimer	specific word, popular movie
indiana jones	specific enough
gothic horror burton	specific words and combination

  - Queries that didn't work as expected:

bomb	expected Oppenheimer, got Dr. Strangelove
toy story	first Finding Nemo, which mentions Toy Story
british western	matches Westerns that won British awards
iron man	first Avengers → makes sense but unexpected
12 years a slave	"12" ignored, rest unspecific, order ignored

Your tutor

- Feedback
  - If you have asked for feedback in your experiences.md, you will find a file feedback-tutor.md in your repository by Friday

    You need to do **svn update** to get it

- Individual help
  - If you have problems or issues that cannot be solved via the forum, you can book a timeslot with your tutor

    See link when you log into Daphne

    You can see who your tutor is by the end of today (Oct 21) or tomorrow (Oct 22) at the very latest + note that your tutor may change over the course of the semester

A glimpse of ES2

- Hands-on practice with rankings and evaluations
  - **Exercise 1:** BM25 scores for better ranking

    BM25 is a formula for word-in-record scores → slides 14 – 17

    Computing these scores is an extension of ES1
  - **Exercise 2:** Write an automatic evaluation script

    Automatically test how well your little "search engine" performs on a given benchmark (queries with their results)
  - **Exercise 3:** Set good parameters for your BM25 scores

    Use the training set we give you for parameter tuning

    Then evaluate once on the test set

    That's also how you do it in "real" research later

Ranking 1/14

- Motivation
  - A keyword query is a formulation of a search desire
  - Not all documents that contain all or some of the keywords are also "relevant" for the search desire

    You already saw this in ES1, and more examples in ES2
  - We want the most "relevant" documents first

    For web search, where we can have millions of hits, this is absolutely crucial for the usefulness of the engine

    But even for the Uni Freiburg web pages, it's non-trivial
  - **Problem:** how to measure how "relevant" a document is?

Ranking 2/14

- Basic Idea

  for now: just SUM

  – In the inverted lists, for each doc id also have a score, which captures how much the document is "about" that word

    university 17 0.5 , 53 0.2 , 97 0.3 , 127 0.8

    freiburg 23 0.1 , 34 0.8 , 53 0.1 , 127 0.7

  – While merging, **aggregate** the scores, then **sort** by score

    MERGED 17 0.5 , 23 0.1 , 34 0.8 , 53 0.3 , 97 0.3 , 127 1.5

    SORTED 127 1.5 , 34 0.8 , 17 0.5 , 53 0.3 , 97 0.3 , 23 0.1

  – The entries in the list are referred to as **postings**

    Above, it's only doc id and score, but a posting can also contain more information, e.g. the position of a word

Ranking 3/14

- Getting the top-k results
  - A full sort of the result list takes time $\Theta(n \cdot \log n)$, where n is the number of postings in the list
  - Typically only the top-k hits need to be displayed
  - Then a **partial sort** is sufficient: get the k largest elements, for a given k

    Can be computed in time $\Theta(n + k \cdot \log k)$

    k rounds of HeapSort yield time $\Theta(n + k \cdot \log n)$

    For constant k these are both $\boldsymbol{\Theta(n)}$

    For ES2, you can ignore this issue

Ranking 4/14

- Meaningful scores
  - How do we precompute good scores

    university 17 0.5 , 53 0.2 , 97 0.3 , 127 0.8

    freiburg 23 0.1 , 34 0.8 , 53 0.1 , 127 0.7
  - **Goal:** the score for the posting for doc D_i in the inverted list for word w should reflect the **relevance** of w in D_i

    In particular, the larger the score, the more relevant
  - **Problem:** relevance is somewhat subjective

    But it has to be done somehow anyway !

Ranking 5/14

- Term frequency (**tf**)
  - The number of times a word occurs in a document
  - **Problem:** some words are frequent in many documents, regardless of the content

university	… , **57** **5** , … … … , **123** **2** , …
of	… , **57** **14** , … … … , **123** **23** , …
freiburg	… , **57** **3** , … … … , **123** **1** , …
SCORE SUM	… , **57** **22** , … … … , **123** **26** , …

  - A word like "of" should not count much for relevance

    Some of you observed that already while trying out queries for ES1

Ranking 6/14

- Document frequency (**df**)
  - The number of documents containing a particular word

    $\mathbf{df}_{\text{university}} = \overset{=2^{14}}{16.384}$, $\mathbf{df}_{\text{of}} = \overset{=2^{19}}{524.288}$, $\mathbf{df}_{\text{freiburg}} = \overset{=2^{10}}{1.024}$

    For simplicity, numbers are powers of 2, see below why

  - Inverse document frequency (**idf**)

    ~~If $df = \frac{N}{2} \Rightarrow idf := 1$~~

    $df = N \Rightarrow idf = 0$, which is FINE

    $\mathbf{idf} = \log_2 (N / df)$ N = total number of documents

    For the example df scores above and $N = 1.048.576 = 2^{20}$

    $\mathbf{idf}_{\text{university}} = 6$ $\left(= \log_2 (2^{20}/2^{14})\right)$, $\mathbf{idf}_{\text{of}} = 1$ (...), $\mathbf{idf}_{\text{freiburg}} = 10$ (...)

    Note: The more frequent a word, the smaller the **idf**

    Without the $\mathbf{log_2}$ small differences in the value of **df** would have too much of an effect

Ranking 7/14

idf university = 6
idf of = 1
idf freiburg = 10

- Combining the two (**tf.idf**)
  - For comparison, here is our earlier **tf** only example

university	… , **57** **5** , … … … , **123** **2** , …
of	… , **57** **14** , … … … , **123** **23** , …
freiburg	… , **57** **3** , … … … , **123** **1** , …
SCORE SUM	… , **57** **22** , … … … , **123** **26** , …

  - And here the same lists with **tf.idf** scores

university	… , **57** **30** (=5·6) , … … … , **123** **12** (=2·6) , …
of	… , **57** **14** (=14·1) , … … … , **123** **23** (=23·1) , …
freiburg	… , **57** **30** (=3·10) , … … … , **123** **10** (=1·10) , …
SCORE SUM	… , **57** **74** , … … … , **123** **45** , …

Ranking 8/14

- Problems with tf.idf in practice
  - The idf part is fine, but the tf part has several problems
  - Let w be a word, and D_1 and D_2 be two documents
  - **Problem 1:** assume that D_1 is longer than D_2

    Then tf for w in D_1 tends to be larger than tf for w in D_2, because D_1 is longer, not because it's more "about" w
  - **Problem 2:** assume that D_1 and D_2 have the same length, and that the tf of w in D_1 is twice the tf of w in D_2

    Then it is reasonable to assume that D_1 is more "about" w than D_2, but just a little more, and not twice more

Ranking 9/14

$tf = 0 \Rightarrow tf^* := 0$

- The **BM25** (best match) formula
  - This tf.idf style formula became famous for outperforming other formulas in standard benchmarks over many years

    BM25 score = tf* · $\underbrace{\log_2 (N / df)}_{= idf}$, where

    $b = 0 \Rightarrow \alpha = 1$

    $b \in [0, 1]$

    tf* = tf · (k + 1) / (k · α + tf) α = (1 – b + b · DL / AVDL)

    tf = term frequency, DL = document length, AVDL = average document length (measured in number of words)
  - Standard setting for **BM25**: k = 1.75 and b = 0.75

    Binary: k = 0, b = 0; Normal tf.idf: k = ∞, b = 0

$$tf^* = tf \cdot \frac{0+1}{0+tf} = 1$$

$$tf^* = tf \cdot \frac{k+1}{k+tf} = tf \cdot \frac{1 + 1/k}{1 + tf/k} \xrightarrow[k \to \infty]{} = tf \cdot \frac{1}{1} = tf$$

Ranking 10/14

- Plausibility argument for BM25, part 1 (tf → tf*)
  - Start with the simple formula tf · idf

    $\alpha = 1$

  - Replace tf by tf* such that the following properties hold:
    - tf* = 0 if and only if tf = 0

      $tf^* = \frac{k+1}{k/tf + 1} > 0$ if $tf > 0$

    - tf* increases as tf increases

      $tf^* = \frac{k+1}{k/tf + 1}$ ✓

    - tf* → fixed limit as tf → ∞

      $tf^* = \frac{k+1}{\underbrace{k/tf}_{\to 0} + 1} \xrightarrow[tf \to \infty]{} k+1$

  - The "simplest" formula with these properties is
    - $tf^* = tf \cdot (k + 1) / (k \cdot \alpha + tf)$ for any α (see next slide)

$$tf^* = tf \cdot \frac{k+1}{k+tf} = \frac{tf \cdot k + tf}{k + tf} = \frac{k+1}{k/tf + 1}$$

Ranking 11/14

- Plausibility argument for BM25, part 2 (choice of α)
  - The α can be understood as a "normalization" of tf

    $tf^* = tf \cdot (k + 1) / (k \cdot \alpha + tf) = tf/\alpha \cdot (k + 1) / (k + tf/\alpha)$
  - BM25 normalizes by the length of the document
    - Full normalization: $\alpha = DL / AVDL$ ... too extreme
    - Some normalization: $\alpha = (1 - b) + b \cdot DL / AVDL$

In the literature, you find much more "theory" behind the BM25 formula. To me, it is **not** more convincing than the simple plausibility arguments from this slide and the last

Ranking 12/14

- BM25 implementation advice
  - First compute the inverted lists with **tf** scores

    We already did that (implicitly) in Lecture 1 (not in ES1)
  - Along with that compute the document length (DL) for each document, and the average document length (AVDL)

    You can measure DL (and AVDL) via the number of words
  - Make a second pass over the inverted lists and replace the **tf** scores by **tf* · idf** scores

    $tf \cdot (k + 1) / (k \cdot (1 - b + b \cdot DL / AVDL) + tf) \cdot \log_2 (N / df)$

    Note that the **df** of a word is just the length (number of postings) in its inverted list

Ranking 13/14

- Further refinements
  - For ES2, play around with the BM25 parameters **k** and **b**
  - Boost results that match each query word at least once

    Warning: when you **restrict** your results to such matches, you miss relevant results, see your bad queries from ES1
  - Somehow take the popularity of a movie into account

    In the file on the Wiki, movies are sorted by popularity

    Note: some scores are already normalized (e.g. IMDb rating), others can't be taken directly as scores (e.g. IMDb #votes)

    A common heuristic to normalize scores with huge differences is to apply the **logarithm** (like we did for computing **idf**)
  - Anything else that comes to your mind and might help …

Ranking 14/14

- Other methods
  - There is a multitude of other sources / approaches for improving the quality of the ranking
  - **Example 1:** Using query logs and click-through data

    Who searches what and clicked on what … important signal in the ranking of big search engines like Google
  - **Example 2:** Learning to rank

    1. For a given query, compute a feature vector (of many different "scores") for a subset of candidate documents
    2. Use machine learning to rank these feature vectors, and thus the documents

Evaluation 1/12

- Ground truth
  - For a set of queries, for each query from that set, determine the set of ids of **all** documents relevant for that query, e.g.

    Query: matrix movies

    Relevant: 10, 582, 877, 10003

  - This set is also called the **ground truth** for that query, and a set of queries + their ground truth is called a **benchmark**
  - For ES2, we have built a benchmark with **56** queries

    Building high-quality benchmarks of sufficient size is an important (and time-consuming) part of research

    

  - On the next slides we will show how to use queries with a ground truth to measure the quality of a search engine

- Precision (**P@k**)
  - The P@k for a given result list for a given query is the percentage of relevant documents among the top-k

Query: matrix movies

Relevant: 10, 582, 877, 10003

REL NOT NOT REL REL NOT

Result list: 582, 17, 5666, 10003, 10, 37, …

REL
877
not in the list

P@1: 1 / 1 = 100% = 1

P@2: 1 / 2 = 50% = 0.5

P@3: 1 / 3 = 33% = 0.33

P@4: 2 / 4 = 50% = 0.5

P@5: 3 / 5 = 60% = 0.6

  - **P@R** = P@k, where k is the number of relevant documents

P@R = P@4 = 50% = 0.5

Evaluation 3/12

- Average Precision (**AP**)
  - Let $R_1, \ldots, R_k$ be the sorted list of positions of the relevant documents in the result list of a given query
  - Then AP is the average of the k $P@R_i$ values

Query: matrix movies

Relevant: 10, 582, 877, 10003

Result list:

REL	NOT	NOT	REL	REL		REL
582,	17,	5666,	10003,	10,	...,	877
1	2	3	4	5		40

$R_1, \ldots, R_4$: 1, 4, 5, 40

$P@R_1, \ldots, P@R_4$: P@1 = 100%, P@4 = 50%, P@5 = 60%, P@40 = $\frac{4}{40}$ = 10%

AP: (100% + 50% + 60% + 10%) / 4 = 55%

Note: AP must also consider documents that do not appear in the result list … for those, we take $P@R_i = 0$

Evaluation 4/12

- Mean Precisions (**MP@k**, **MP@R**, **MAP**)
  - Given a benchmark with several queries + ground truth
  - Then one can capture the quality of a system by taking the **mean** (average) of a given measure over all queries

    MP@k = mean of the P@k values over all queries

    MP@R = mean of the P@R values over all queries

    MAP = mean of the AP values over all queries

    These are very common measures, which you will find in many research papers concerned with ranking objects

Evaluation 5/12

- Discounted Cumulative Gain (DCG, nDCG)
  - Sometimes relevance comes in more than one shade, e.g.

    0 = not relevant, 1 = somewhat rel, 2 = very relevant
  - There should be a <u>bonus</u> if very relevant documents are ranked higher than only somewhat relevant documents
  - **Cumulative gain:** $CG@k = \Sigma_{i=1..k}\, rel_i$

    rel_i = relevance of i-th doc in result list
  - **Discounted CG:** $DCG@k = \Sigma_{i=1..k}\, rel_i \,/\, \log_2 (i+1)$
  - Problem: CG and DCG are not normalized to [0,1]
  - Solution: normalize by maximally achievable value
  - **Ideal DCG:** iDCG@k = DCG@k of ideal ranking
  - **Normalized DCG:** nDCG@k = DCG@k / iDCG@k

Evaluation 6/12

- Discounted Cumulative Gain (DCG, nDCG), example
  - Consider the following result list and relevances, assuming only 3 relevant documents overall (for this query)

Hit #1:	relevant	1
Hit #2:	not relevant	0
Hit #3:	very relevant	2
Hit #4:	relevant	1
Hit #5:	not relevant	0

$\mathrm{DCG@k} = \sum_{i=1}^{k} rel_i / \log_2(i+1)$

  - Then **DCG@5** = $\underbrace{\frac{1}{\log_2 2}}_{=1} + \frac{0}{\log_2 3} + \underbrace{\frac{2}{\log_2 4}}_{=2} + \underbrace{\frac{1}{\log_2 5}}_{=2.34\ldots} + \frac{0}{\log_2 6}$

$= 1 + 1 + \frac{1}{\log_2 5} = 2.42\ldots$

  - And **iDCG@5** = $\frac{2}{\log_2 2} + \frac{1}{\log_2 3} + \frac{1}{\log_2 4} + \frac{0}{\log_2 5} + \frac{0}{\log_2 6}$

$= 2 + \underbrace{\frac{1}{\log_2 3}}_{=0.63} + 0.5 = 3.13$

  - Hence **nDGC@5** = $\frac{2.42\ldots}{3.13\ldots} \approx 77\%$

A common approach in competitions is **pooling**: jugde only the top-k results from each participant

- Binary preference (bpref)
  - Sometimes we have relevance judgements only for a subset of all the documents

    Typically for very large datasets, where it is infeasible to check all the documents for relevance

  - Then for each judged relevant doc, compute a score based on how many judged non-relevant docs come before it, and take **bpref** as the average of these scores, formally:

    $\text{bpref} = 1/|R| \cdot \Sigma_{d \in RR} (1 - |NR(d)| / \min(|R|, |N|))$ where

    R = docs judged relevant
    N = docs judged non-relevant
    RR = docs judged relevant in result list
    $NR(d)$ = docs from the $|R|$ top-ranked non-relevant before d

- Binary preference (bpref), example
  - Consider the following result list

#1: ???
#2: not relevant
#3: relevant
#4: not relevant
#5: ???
#6: ???
#7: not relevant
#8: relevant
#9: ???

Not in list:

One more relevant document
Ten more non-relevant documents

$R = \{\#3, \#8\}$

$|R| = 3$

$|N| = 13$

$\min\{|R|, |N|\} = 3$

$NR(\#3) = \cancel{1}\ \{\#2\}$

$NR(\#8) = \cancel{3}\ \{\#2, \#4, \#7\}$

$$1 - \frac{|NR(\#3)|}{\min(|R|,|N|)} = 1 - \frac{1}{3} = 2/3$$

$$1 - \frac{|NR(\#8)|}{\min(|R|,|N|)} = 1 - \frac{3}{3} = 0$$

$$BPREF = \frac{2/3 + 0 \cancel{+ 0}}{3} = \frac{2}{9} = 22\%$$

Evaluation 9/12

- Binary preference (bpref), finer points
  - Why $NR(d) / \min(|R|, |N|)$ and not $NR(d) / |N|$?

    First verify that both $NR(d) \le |R|$ and $NR(d) \le |N|$, so that $NR(d) / \min(|R|, |N|)$ is always in the range $[0, 1]$

    Then note that $1 - NR(d) / \min(|R|, |N|)$ becomes **zero** already when only $|R|$ non-relevant docs come before d

    With $NR(d) / \min(|R|, |N|)$ instead of $NR(d) / |N|)$ it thus becomes harder to achieve a high **bpref** score
  - Why $1 / |R|$ and not $1 / |RR|$?

    It should be punished if a relevant doc is not in the result list

    It's the same reason that for AP (average precision) on slide 23, we took $P@R_i = 0$ for docs that are not in the result list

Evaluation 10/12

- Overfitting
  - Tuning parameters / methods to achieve good results on a given benchmark is called **overfitting**

    In an extreme case: for each query from the benchmark, output the list of relevant docs from the ground truth
  - In a realistic environment (real search engine or competition), one is given a **training** set for development

    The actual evaluation is on a **test** set, which must not be used or is not available during development

    For ES2, we explicitly provide a training benchmark for development, and a test benchmark to measure the quality of your system once you are done with tuning

Evaluation 11/12

- Set vs. ranked list
  - On the previous slides, the ground truth for each query was a **set** and the results we evaluated were **ranked lists**

    That looks strange, why didn't we evaluate it in one of the following three ways instead?

  - **1. Why** don't we let our search engine also output a set and measure how similar it is to the ground truth?

    Because there is no natural cut-off value

  - **2. Why** isn't our ground truth also a ranked list and we measure the similarity of the two rankings?

    Because the set of relevant documents usually doesn't have a full order (though there might be levels of relevance)

Evaluation 12/12

- Set vs. ranked list, continued
  - **3. Why** don't we evaluate the accuracy of the scores themselves, instead of only the order that is implied by these scores?

    The absolute values of our scores don't mean much, we only compute them as a means to rank the documents

    Note: there are indeed problems, where the goal is to predict an absolute score as accurately as possible

    These are called regression problems, and we will encounter them in a later lecture

References

- In the Manning/Raghavan/Schütze textbook

  Section 6: Scoring and Term Weighting

  Section 8: Evaluation in Information Retrieval

- Relevant Papers

  Probabilistic Relevance: BM25 and Beyond **FnTIR 2009**

  Test Collection Based Evaluation of IR Systems **FnTIR 2010**

- Relevant Wikipedia articles

  http://en.wikipedia.org/wiki/Okapi_BM25

  https://en.wikipedia.org/wiki/Information_retrieval#Performance_and_correctness_measures