Information Retrieval

WS 2017 / 2018

Lecture 2, Tuesday October 24th, 2017

(Ranking, Evaluation)

Prof. Dr. Hannah Bast
Chair of Algorithms and Data Structures
Department of Computer Science
University of Freiburg

Overview of this lecture

- Organizational
  - Your experiences with ES1 Inverted index
  - No lecture next week Thanks Martin Luther
- Contents
  - Ranking tf.idf and BM25, other tricks
  - Evaluation Ground truth, Precision, AP, MAP, nDCG, BPref, Overfitting
  - **ES2:** implement BM25, tune your ranking, and then evaluate on a small benchmark

    There will be a small competition (table on the Wiki)

Experiences with ES1 1/3

- Summary / excerpts
  - For some of you, it took some time to "get started"

    Linux, SVN, out of programming practice, …
  - The task itself was (deliberately) easy and nice

    Quite a few implemented extra features, like highlighting
  - Suggestions: GIT instead of SVN, terminal recording

    GIT has no access control, video recording not sufficient?
  - Why is it called "inverted" index?

    The movies.txt provides the words for each record, the inverted index provides the records for each word, so in this sense, it's an "inverse" representation of the data

Experiences with ES1   2/3

- Results
  - Examples for queries that work well:

james bond	sufficiently specific keywords, works well even without ranking
guadalquivir	maximally specific keyword

  - Examples for queries that don't work well

titanic 1997	numbers not indexed
NCIS	series not in movies.txt
space balls	in movies.txt it's spaceballs
forest gump	typo, it's forrest gump
redemption	The Shawshank Redemption listed late (at line 284) in movies.txt

Experiences with ES1 3/3

- Linear construction time?
  - Quite a few of you implemented something like this:

```
for line in file:
    ...
    if record_id not in self.inverted_lists[word]:
        self.inverted_lists[word].append(record_id)
```

  - Then index construction on movies.txt takes **very** long:

the "not in" takes linear time, not constant time

which means the whole loop take **quadratic time**

Super-important piece of advice: **never** use built-in functions without understanding their time-complexity

Ranking 1/14

- Motivation
  - Queries often return many hits
  - Typically more than one wants to (or even can) look at

    For web search: often millions of documents

    But even for less hits a proper ranking is **key** to usability
  - So we want to have the most "relevant" hits first
  - **Problem:** how to measure what is how "relevant"

Ranking 2/14

here : SUM

- Basic Idea
  - In the inverted lists, for each doc id also have a score

    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

    $\mathbf{df}_{\text{university}} = 16.384$, $\mathbf{df}_{\text{of}} = 524.288$, $\mathbf{df}_{\text{freiburg}} = 1.024$

    For simplicity, number are powers of 2, see below why
  - Inverse document frequency (**idf**)

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

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

    $\mathbf{idf}_{\text{university}} = 6$, $\mathbf{idf}_{\text{of}} = 1$, $\mathbf{idf}_{\text{freiburg}} = 10$

    Understand: 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**)
  - Reconsider 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** , …

  - Now combined with **idf** scores from previous slide

university	… , **57** **30** , … … … , **123** **12** , …
of	… , **57** **14** , … … … , **123** **23** , …
freiburg	… , **57** **30** , … … … , **123** **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 then 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

- The **BM25** (best match) formula
  - This tf.idf style formula has consistently outperformed other formulas in standard benchmarks over the years

    BM25 score = tf* · $\log_2$ (N / df), where

    $$\textbf{tf*} = \text{tf} \cdot (k + 1) / (k \cdot \underbrace{(1 - b + b \cdot DL / AVDL)}_{\alpha} + \text{tf})$$

    tf = term frequency, DL = document length, AVDL = average document length
  - Standard setting for **BM25**: k = 1.75 and b = 0.75

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

$b = 0 \Rightarrow \alpha = 1$

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

$$= \frac{tf}{tf} = 1 \quad ☒$$

$b = 0 \Rightarrow \alpha = 1$

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

$$= tf \cdot \frac{1 + 1/k}{1 + tf/k} \xrightarrow{k \to \infty} tf \quad ☒$$

Ranking 10/14

- Plausibility argument for BM25, part 1
  - Start with the simple formula tf · idf
  - Replace tf by tf* such that the following properties hold:
    - tf* = 0 if and only if tf = 0 $\quad tf \cdot \frac{k+1}{k+tf} = 0 \iff tf = 0 \quad \checkmark$
    - tf* increases as tf increases $\quad tf^* = \frac{k+1}{k/tf + 1} \quad \checkmark$
    - tf* → fixed limit as tf → ∞ $\quad tf^* = \frac{k+1}{k/tf + 1} \xrightarrow{tf \to \infty} k+1 \quad \checkmark$
  - The "simplest" formula with these properties is
    - tf* = tf · (k + 1) / (k + tf) $= \frac{tf \cdot (k+1)}{k+tf} = \frac{k+1}{k/tf + 1}$

Ranking 11/14

- Plausibility argument for BM25, part 2
  - So far, we have tf* = tf · (k + 1) / (k + tf)
  - Normalize by the length of the document
    - Replace tf by tf / α
    - Full normalization: α = DL / AVDL … too extreme
    - Some normalization: α = (1 – b) + b · DL / AVDL
  - This gives us tf* = tf / α · (k + 1) / (k + tf / α)
  - And hence tf* = tf · (k + 1) / (k · α + tf)
  - The final BM25 score is then tf* · idf

no normalization

↑

b=0 : 1

b=1 : DL/AVDL

↑

full normalization

Lots of "theory" behind this formula, but to me not really more convincing than these simple plausibility arguments

Ranking 12/14

- Implementation advice
  - First compute the inverted lists with **tf** scores

    We already did that (implicitly) in Lecture 1
  - 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

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

    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 might miss some relevant results

    For example: steven spielberg **movies**
  - Somehow take the popularity of a movie into account

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

    Popularity scores also have a Zipf distribution, so you might take ~ $\mathbf{N}^{-\alpha}$ as popularity score for the N-th movie in the list
  - Anything else that comes to your mind and might help …

Ranking 14/14

- Advanced 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 … main pillar for the result quality of big search engines like Google
  - **Example 2:** Learning to rank

    Using machine learning (more in a later lecture) to find the best parameter settings in ranking function

Evaluation 1/9

- Ground truth
  - For a given query, the ids of all documents relevant for that query

Query:	matrix movies
Relevant:	10, 582, 877, 10003

  - For ES2, we have built a ground truth for 10 queries

    Building a good and large enough ground truth is a common (and time-consuming) part in research in IR

Evaluation 2/9

- 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		REL
Result list:	582,	17,	5666,	10003,	10,	37,	…	877
	1.	2.	3.	4.	5.	6.		

P@1: 1/1 = 100%

P@2: 1/2 = 50%

P@3: 1/3 = 33%

P@4: 2/4 = 50% ← P@R

P@5: 3/5 = 60%

- **P@R** = P@k, where k = #relevant documents

Evaluation 3/9

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

Query:	matrix movies
Relevant:	10, 582, 877, 10003
Result list:	$\underset{1.}{\overset{\text{REL}}{582}}, \underset{2.}{\overset{\text{NOT}}{17}}, \underset{3.}{\overset{\text{NOT}}{5666}}, \underset{4.}{\overset{\text{REL}}{10003}}, \underset{5.}{\overset{\text{REL}}{10}}, \ldots, \underset{40.}{\overset{\text{REL}}{877}}$
$R_1, \ldots, R_4$:	1, 4, 5, 40
$P@R_1, \ldots, P@R_4$:	100% , 50% , 60% , 10%
AP:	(100% + 50% + 60% + 10%) / 4 = 55%

Note: for documents not in result list, just take $P@R_i = 0$

Evaluation 4/9

- 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 a lot of research papers on information retrieval

Evaluation 5/9

rel_i = relevance of i-th doc in result list

- 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 "discount" in the score if very relevant documents come after only somewhat relevant documents
  - **Cumulative gain:** $\text{CG@k} = \Sigma_{i=1..k} \text{rel}_i$
  - **Discounted CG:** $\text{DCG@k} = \text{rel}_1 + \Sigma_{i=2..k} \text{rel}_i / \log_2 i$
  - Problem: CG and DCG are larger for larger result lists
  - Solution: normalize by maximally achievable value
  - **Ideal DCG:** iDCG@k = DCG@k of ideal ranking
  - **Normalized DCG:** nDCG@k = DCG@k / iDCG@k

Evaluation 6/9

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

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

the „discount" for position 4

  - Then **DCG@5** = $2 + \frac{1}{\log_2 2} + \frac{2}{\log_2 4} = 2 + 1 + 1 = 4$
  - And **iDCG@5** = $2 + \frac{2}{\log_2 2} + \frac{1}{\log_2 3} = 2 + 2 + 0.63 = 4.63$
  - Hence **nDGC@5** = $4 / 4.63 = 0.86 = 86\%$

Evaluation 7/9

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 measure whether judged relevant documents come before judged non-relevant documents
  - $\text{bpref} = 1/|R| \cdot \Sigma_{r \in RR} (1 - |NR(r)| / \min(|R|, |N|))$ where

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

Evaluation 8/9

R = all docs judged relevant
N = — " — not relevant
RR = docs in R and result list

- Binary preference (bpref), example
  - Consider the following result list and relevances, assuming 7 judged documents overall for a particular query

#1: judged relevant
#2: not judged
#3: not judged
#4: judged not relevant
#5: judged relevant

Not in result list:

one doc (A) judged relevant
three docs (X,Y,Z) judged not relevant

$\min\{\overset{3}{|R|}, \overset{4}{|N|}\} = 3$

- R = {#1, #5, A}, N = {#4, X, Y, Z}, RR = {#1, #5}
- NR(Hit #1) = $\emptyset$, $|NR(\#1)| = 0$
- NR(Hit #5) = #4 $|NR(\#5)| = 1$
- bpref = $\frac{1}{3}\Big(\underbrace{\left(1 - \frac{0}{3}\right)}_{\text{for \#1}} + \underbrace{\left(1 - \frac{1}{3}\right)}_{\text{for \#5}}\Big) = \frac{1 + \frac{2}{3}}{3} = \frac{5}{9}$ $= 55{,}5\%$

Evaluation 9/9

- 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 / was not available during development

    For ES2, do the development / tuning on some queries of your choice, then evaluate without further changes

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