Information Retrieval

WS 2015 / 2016

Lecture 2, Tuesday October 27th, 2015
(Ranking, Evaluation)

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

UNI FREIBURG

Overview of this lecture

- Organizational
  - Your experiences with ES1 — Inverted index
  - Choice of language — Python, Java, C++
- Contents
  - Ranking — tf.idf and BM25
  - Evaluation — Ground truth, Precision, …

  - **Exercise Sheet #2:** implement BM25, tune your ranking, and then evaluate on a small benchmark

    There will be small competition (table on the Wiki)

Experiences with ES1 1/3

- Summary / excerpts
  - Nice and interesting exercise
  - Easy for those with sufficient programming experience
  - More work than expected for those out of practice

    "First I thought the task is very easy and I'm finished soon, but then I got stuck in small coding details."

    Don't worry, this will get much better over the course
  - Getting used to Daphne and SVN took some time

    But many of you are used to it already

Experiences with ES1 2/3

- Results
  - Queries with **only** specific words work very well

    matrix , hobbit , edward scissorhands , …
  - Already one **unspecific** word often leads to bad results

    the matrix , **tom** hanks , lord **of the** rings , …
  - Several of you implemented cool additional features

    Highlighting, better ranking, command history (wow), autocompletion (wow wow wow), …

Experiences with ES1 3/3

- Choice of programming language
  - Let me briefly repeat what I said in Lecture 1

    Python will be the least work

    Feel absolutely free to also use Java or C++, but be prepared for more work

    For ES1, we provided equivalent code for **all three** languages, to give you a working example for each

    We will not do that for future lectures, except:

    For lectures about efficiency, we will provide code in both Java and C++, but not in Python

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

Score : ADD

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

    uni 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

    uni   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

Many of you observed that already, working on 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 **df** would have too much of an effect ; why exactly $\mathbf{log_2}$ → later slide

Ranking 7/14

- 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

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

    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 $\Rightarrow \alpha = 1$

$$tf^* = tf \cdot (0+1) / (0 \cdot \ldots + tf) = 1$$

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

- 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
    - tf* increases as tf increases
    - tf* → some limit as tf → ∞
  - The simplest formula with these properties is
    - tf* = tf · (k + 1) / (k + tf)

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

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

$tf^* \xrightarrow[tf \to \infty]{} k+1$

Ranking 11/14

$b = 0 \Rightarrow \alpha = 1$ no norm.
$b = 1 \Rightarrow \alpha = DL / AVDL$ full norm.

- Plausibility argument for BM25, part 2
  - So far, we have tf* = tf · (k + 1) / (k + tf)
  - Normalize by the length of the document
    - full normalization: α = DL / AVDL ... too extreme
    - some normalization: α = (1 – b) + b · DL / AVDL
    - replace tf by tf / α
  - This gives us tf* = tf / α · (k + 1) / (k + tf / α)
  - And hence tf* = tf · (k + 1) / (k · α + tf)

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

Ranking 12/14

number of words is fine

UNI FREIBURG

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

    You already did that (implicitly or explicitly) for ES1
  - Along with that compute the document length (DL) for each document, and the average document length (AVDL)
  - 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 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, for example:

    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

    Learning to rank

    Using machine learning (more about that in a later lecture) to find the best parameter setting

Evaluation 1/6

for ES2, use movies2.txt

- 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

    That was a lot of work, mostly Björn's, **thanks !**

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

Evaluation 2/6

P@R = P@r, where
r = # relevant documents

- 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 REL, NOT REL, REL, REL, NOT REL
Result list:	582, 17, 5666, 10003, 10, 37, …
P@1:	1/1 = 100%
P@2:	1/2 = 50%
P@3:	1/3 = 33%
P@4:	2/4 = 50% = P@R for this query
P@5:	3/5 = 60%

Evaluation 3/6

- 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
	REL, NOT REL, NOT REL, REL, REL, NOT REL, REL
Result list:	582, 17, 5666, 10003, 10, …, 877, …
	1., 2., 3., 4., 5., 40.
$R_1, \ldots, R_4$:	1, 4, 5, 40
$P@R_1, \ldots, P@R_4$:	100%, 50%, 60%, 10%
AP:	55%

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

Evaluation 4/6

- 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

Evaluation 5/6

- Other measures
  - There is a BIG variety of other evaluation measures, e.g.
  - **nDCG** = normalized discounted cumulative growth

    Takes into account that documents can have varying degrees of relevance, e.g. primary and secondary

    Gives credit if primary is ranked before secondary
  - **BPref** = binary relevance … preference relation

    Takes into accounts that some documents are unjudged

    This is a frequent problem in benchmarks for huge text corpora, where complete judgment is impossible

    E.g. all relevant document for "tom hanks" on the web

Evaluation 6/6

- 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 on 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