Information Retrieval

WS 2013 / 2014

Lecture 2, Tuesday October 29th, 2013

(Ranking, tf.idf, BM25, Vector Space Model, 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 Ex. Sheet 1 (inverted index)
- How to rank results
  - Basic principle / scores
  - Formulas: tf.idf and BM25
  - Vector Space Model
  - Quality evaluation: precision, recall, …, nDCG@k

Exercise Sheet #2: compare three ranking formulas with respect to their nDCG@5 for query of your choice

Experiences with ES1 (inverted index)

- Summary / excerpts last checked October 29, 15:00
  - Liked the style of the lecture / exercises
  - Heard stuff about SVN etc. for the 100^{th} time
  - No major problems for most, good exercise for starters
  - Some overhead for setting up the environment
  - Easy to overlook the implementation note in .TIP file
  - The usual complaints about the style checker

    Tabs vs. spaces, how to place the { … }
  - Put slides and exercise sheet in the SVN, too

    Not so easy as it sounds, but I try to find a way …
  - "Unfortunately couldn't finish, but it was fun."

Ranking 1/4

- 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, recall the Broccoli demo from Lecture 1
  - So we want to have the most "relevant" hits first
  - Problem: how to measure what is how "relevant"

Ranking 2/4

- 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
  - When intersecting lists aggregate (here: **add**) the scores

    uni freiburg 53 0.3, 127 1.5
  - Then sort the result by score

    uni freiburg 127 1.5, 53 0.3
  - 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 3/4

- Generalization
  - We can do the same thing with computing the **union**

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
UNION	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

  - Note: documents which contain only some (or one) of the words can be ranked before documents containing all of the words

    provided the individual scores are high enough

  - This is also called **and-ish** retrieval … like AND, but not exactly

    For ES2 you can continue to use intersection

Ranking 4/4

- Getting the top-k results
  - A full sort takes time $\Theta(n \cdot \log n)$, where n = #documents
  - 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

    This 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 $\Theta(n)$
  - In C++ there is std::sort and std::partial_sort
  - In Java there is Collections.sort but no partial sort method

For ES2, you can but don't have to use partial sort

Scores 1/8

- How to compute meaningful scores
  - Let S_1, S_2, S_3, … be the score sums of the documents D_1, D_2, D_3, … for a given keyword query Q
  - **GOAL:** S_i should reflect the relevance of D_i for Q

    in particular: $S_i > S_j \rightarrow D_i$ more relevant for Q than D_j
  - Obviously a very hard problem

    In particular, it is often less than clear what is the search request behind a given query

    For example: freiburg doctor
  - But it has to be done anyway !

Scores 2/8

- One important factor: tf = term frequency

  tf of a word w in a doc D = how often w occurs in D

  – Problem with mere tf scores: some words are frequent in many documents, regardless of content

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

  But the **tf score** for "of" should not count that much for relevance

Scores 3/8

- Another important factor: **df** = document frequency

  df of a word w = the number of docs containing w

  – For example ... for simplicity, numbers will be powers of 2

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

  – Intuitively, words with a large df should not count as much; thus consider the inverse document frequency

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

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

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

Scores 4/8

- Combining the two: **tf.idf** = tf · idf = tf · $\log_2$ (N / df)
  - 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** , …

Scores 5/8

- 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** (example)

    If D_1 is longer than D_2, it will tend to have a higher tf for w already because it's longer, not because it's more "about" w
  - **Problem 2** (example)

    If D_1 and D_2 have the same length, and 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

Scores 6/8

$k=0,\ b=0:\ tf^* = tf / tf = 1$

$k=\infty,\ b=0:\ tf^* = \lim_{k\to\infty} \frac{tf\cdot(k+1)}{k+tf} = \lim_{k\to\infty} \frac{tf(1+\frac{1}{k})}{1+\frac{tf}{k}} = tf$

- **BM25** = Best Match 25, Okapi = an IR system
  - This tf.idf style formula has consistently outperformed other formulas in standard benchmarks over the years

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

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

    For ES2, you can just take #Bytes

    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
  - There is "theory" behind this formula ... see references
  - Next slide: simple reason why the formula makes sense

Scores 7/8

- Why BM25 makes sense
  - Start with the simple formula tf · idf
  - Replace tf by tf* = tf · (k + 1) / (k + tf)
    - tf* = 0 if and only if tf = 0
    - tf* increases as tf increases
    - tf* → k + 1 as tf → infinity
  - Normalize by the length of the document
    - full normalization: alpha = DL / AVDL
    - some normalization: alpha = (1 – b) + b · DL / AVDL
    - replace tf by tf / alpha' (in the formula for tf* above)

the same argument goes for the $\log_2$ in idf.

BM25 is the simplest formula with these properties!

e.g. $\alpha = 2$, $b = 0.5$
$\Rightarrow \alpha' = 0.5 + 0.5 \cdot 2 = 1.5$

for $b = 0 \Rightarrow \alpha' = 1$

Scores 8/8

- Implementation advice
  - The entries in the inverted lists are now elements of a class Posting, each holding a doc id and a score

    ```
    Map<String, Array<Posting>> invertedLists;
    ```

  - During parsing, compute only basic tf: when a document contains a word multiple times, simply add 1 to the score
  - After the parsing, the length of each inverted list is exactly the df for that word, which also gives you the idf then

    The final tf.idf / BM25 scores can then be obtained by another pass over each of the inverted lists

    See the TIP files for ES2 linked on the Wiki

Vector Space Model 1/4

- Basic Idea
  - View documents (and queries) as **vectors** in a vector space
  - Each dimension corresponds to a word from the vocabulary
  - Entries can be according to any of our scoring formulas
  - Here is an example

in the example below: binary scores

Document 1: university of freiburg
Document 2: university of karlsruhe
Document 3: freiburg cathedral

QUERY: university freiburg

	Doc 1	Doc 2	Doc 3	QUERY
university	1	1	0	1
freiburg	1	0	1	1
karlsruhe	0	1	0	0
of	1	1	0	0
cathedral	0	0	1	0

Vector Space Model 2/4

$d_1 \; d_2$: $(1,1,0,1)$ and $(2,2,0,2)$ $\Rightarrow \cos \angle(d_1, d_2) = 1$ (0°)

- Similarity between two documents

  or between a document and a query

  $d_1 \; d_2$: $(1,1,0,1)$ and $(0,0,1,0)$ $\Rightarrow \cos \angle(d_1, d_2) = 0$ (90°)

  - **Cosine similarity:** $\text{sim}(d_1, d_2) = \cos \text{angle}(d_1, d_2)$

    This is **1** if $d_1 \sim d_2$, and **0** if no word in common

    Advantage: favorable properties for mathematical analysis

  - **Dot product:** d1 • d2 = sum of products of components

    Advantage: easy to compute efficiently ... later slide

  - From linear algebra: $d_1 \bullet d_2 = |d_1| \cdot |d_2| \cdot \cos \text{angle}(d_1, d_2)$

  - Therefore, if the vectors are length normalized ($|\cdot| = 1$) then

    dot product = cosine similarity

$$\begin{pmatrix}1\\1\\0\\1\end{pmatrix} \bullet \begin{pmatrix}1\\2\\0\\0\end{pmatrix} = 1\cdot1 + 1\cdot2 + 0\cdot0 + 1\cdot0 = 3$$

Vector Space Model 3/4

- Computing the dot product similarity

  … of a query to all documents

  – Mathematically, this is just a multiplication of the term-document matrix A with the query q

  $$A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (5 \times 3) \qquad q = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

  – The straightforward algorithm needs time $\Theta(m \cdot n)$, where m = total num. of words, n = num. of documents

  – However: A is a very **sparse** matrix, and q is a very **sparse** vector, where sparse = most entries are zero

$$A^T \cdot q = \begin{matrix} \text{Doc1} \\ \text{Doc2} \\ \text{Doc3} \end{matrix} \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}$$

$3 \times 5 \qquad 5 \times 1 \qquad 3 \times 1$

these are the dot-product similarities of q and our three documents

Vector Space Model 4/4

- Computing the dot product similarity
  - Observation 1: Inverted lists are nothing else, but a representation of the non-zero entries of the term-document matrix
  - Observation 2: Computing the dot product of a query Q with **every** document is nothing else but:

    Taking the union of the inverted lists of all words in Q with a non-zero entry and adding up the scores accordingly

Evaluation 1/6

- How to evaluate the quality of a ranking
  - **Variant 1:** For each query, identify the **ground truth** = all relevant documents for that query

    This is a very time-consuming job, especially for large document collections. But once done, easy + quick re-evaluation after any changes / tuning to your system

    For big data, use services like Amazon's Mechanical Turk
  - **Variant 2:** For each query, manually inspect the result list for relevant documents

    For ES2, just do a manual inspection of the top-5 hits
  - **Variant 3:** In competitions, pooling is sometimes used = manually evaluate only the union of the top-k hits from all participating systems, where e.g. k = 100

Evaluation 2/6

Relevant docs: 17, 45, 51, 107
My search found: 17, 51, 90

- Precision and Recall, ranking-unaware measures
  - Let tp = the number of relevant docs in the result list (true positives)
  - Let fp = the number of non-relevant docs in the result list (false positives)
  - Let fn = the number of relevant docs missing from the result list (false negatives)
  - Then **precision** is defined as tp / (tp + fp)

    and **recall** is defined as tp / (tp + fn)
  - **F-measure** = harmonic mean of the two

prec = $\frac{2}{3}$ = 67%
rec. = $\frac{2}{4}$ = 50%

Evaluation 3/6

Search returned:
1. Doc57 ✓
2. Doc12 ✓
3. Doc107 ✗
4. Doc13 ✓
5. Doc14 ✗

Prec@5 = 60%

- Precision and Recall, ranking-aware measures
  - Precision@k = the precision among the first k docs
  - Precision@R = the precision among the first R docs

    where R is the number of relevant documents
  - Let $k_1 < \ldots < k_R$ be the ranks of the relevant docs in the result list (rank missing docs randomly or worst-case)

    Average precision = average of $P@k_1, \ldots, P@k_R$
  - For a set of queries, the **MAP** = mean average precision is the average (over all queries) of the average precisions

Evaluation 4/6

- Precision-recall curve
  - Average precision is just a single number
  - For a complete picture of the quality of the ranking, plot a precision-recall curve
  - If the x-axis is normalized, these can also be averaged over several queries

Evaluation 5/6

- More refined measures
  - Sometimes relevance comes in more than one shade, e.g.

    0 = not relevant, 1 = somewhat rel, 2 = very relevant
  - Then a ranking that puts the very relevant docs at the top should be preferred

    Cumulative gain $CG@k = \Sigma_{i=1..k}\ rel_i$

    Discounted CG $DCG@k = rel_1 + \Sigma_{i=2..k}\ rel_i / \log_2 i$
  - **Problem:** CG and DCG are larger for larger result lists
  - **Solution:** normalize by maximally achievable value

    iDCG@k = value of DCG@k for **i**deal ranking

    Normalized DCG nDCG@k = DCG@k / iDCG@k

Evaluation 6/6

- Normalized discounted cumulative gain, example

1. very relevant 2
2. relevant 1
3. not relevant 0
4. very relevant 2
5. not relevant 0

$$DCG@5 = 2 + \frac{1}{\log_2 2} + \frac{0}{\log_2 3} + \frac{2}{\log_2 4} + \frac{0}{\log_2 5}$$

$$= 2 + 1 + \frac{2}{2} = 4$$

$$iDCG@5 = 2 + \frac{2}{\log_2 2} + \frac{2}{\log_2 3} + \frac{2}{\log_2 4} + \frac{2}{\log_2 5}$$

References

- In the Raghavan/Manning/Schütze textbook

  Section 6: Scoring, term weighting, vector space model

- Relevant Papers

  The Probabilistic Relevance Framework: BM25 and Beyond

  S. Robertson and H. Zaragoza FnTIR 2009, 333 – 389

- TREC conference (benchmarks)

  http://trec.nist.gov/tracks.html

- Relevant Wikipedia articles

  http://en.wikipedia.org/wiki/Okapi_BM25
  http://en.wikipedia.org/wiki/Precision_and_recall
  http://en.wikipedia.org/wiki/Discounted_cumulative_gain
  http://en.wikipedia.org/wiki/Partial_sorting