Search Engines

WS 2009 / 2010

Lecture 3, Thursday November 5th, 2009

(Ranking, Tf.idf, BM25, Precision, Recall, Top-K)

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

Why Ranking

- Very simple answer:
  - for almost any query on any collection you will get a lot of hits
  - it's a lot of work or impossible to look at all of them
  - therefore it's important that those hits which are most relevant for your query are shown first
  - Give example of a Google query
  - Note: Google's ranking formula is like the Coca-Cola recipe

Goals for Lecture 3

- Learn essential things about ranking (of search results)
  - how to assign scores to documents wrt a query
  - state of the art formulas for such scores
  - understand the rationale behind these formulas
  - how to measure whether a ranking is good or bad
  - how to compute the top-ranked documents efficiently (without looking at all possible hits)
- Exercises
  - practical: analyze a query of your choice
  - theoretical: some things about scoring

Two ways of ranking

- Query-independent
  - alphabetically
  - by date of creation (e.g. for a publication)
  - by date of last modification (e.g. for files)
  - ...
  - very easy to compute
- Query-dependent
  - somehow measure the relevance of each document to the query
  - how does one make relevance objective?

Query-dependent Ranking

- Some factors that should influence ranking
  - the more often a document contains a query word the higher it should get ranked
  - some words (or: terms) are more "significant" than others

    for example: informatik versus and
  - intuitively, the most significant terms in a document are those
    - which are not very frequent overall
    - but occur frequently in this particular document
  - this is exactly what tf.idf is trying to address
    - see next slide

tf.idf

- tf = term frequency
  - the term frequency of a term in a document is the number of occurrences of that term in that document
- idf = inverse document frequency
  - the df = document frequency of a term is the number of documents containing that term
  - the inverse document frequency is a function that decreases as the document frequency increases

    for example idf = log (N / df), where N = # all documents
- tf.idf is simply the product of the two
  - for example tf.idf = tf · log (N / df)

Note: there are many tf.idf formulas, not just one

How to rank from tf.idf scores

- Easy:
  - inverted lists with scores (produced by parser)

informatik	Doc12	Doc57	Doc59	Doc61	Doc77	...
	0.2	0.7	0.3	0.3	0.9	...
freiburg	Doc5	Doc12	Doc13	Doc14	Doc67	...
	0.2	0.4	0.3	0.3	0.9	...

  - form *union* (not intersection) of lists, computing the sum of the scores for each document

informatik	Doc5	Doc12	Doc57	Doc61	Doc77	...
freiburg	0.2	0.6	0.7	0.3	0.9	...

Note: document containing not all query words may rank high

sometimes called “and-ish” retrieval

Vector Space Model

- Term-document matrix (with tf.idf scores)

	Doc1	Doc2	Doc3	Doc4	Doc5
internet	0.9	0.0	0	0.6	0
web	0.3	0.9	0	0.4	0
surfing	0.7	0.6	0	0.8	0.6
beach	0	0	1.0	0.3	0.7

Qry
1.0
0
0
0

- Similarity between docs = similarity between vectors
  - e.g., just take scalar product
    - high if vectors have many terms in common
    - zero if vectors have no terms in common
- Note: query can be viewed as a vector in the same way

Vector Space Model

- Term-document matrix (with tf.idf scores)

	Doc1	Doc2	Doc3	Doc4	Doc5
internet	0.9	0.0	0	0.6	0
web	0.3	0.9	0	0.4	0
surfing	0.7	0.6	0	0.8	0.6
beach	0	0	1.0	0.3	0.7

Qry
1.0
0
0
0

- Cosine similarity
  - cosine of angle between vectors
  - this is equal to the scalar product if the vectors are normalized
  - advantage: deals with different document lengths
    - see example on next slide

Vector Space Model — Document Length

- Give example query and two documents
  - query contains exactly one word
  - both documents contain the word the same number of times
  - but one is much longer than the other
  - the shorter one should be preferred

Relevance Feedback

- Assume the user says which of the documents returned are relevant
  - then take these documents and form their "average"
  - take this average document as a new query
  - rank documents according to this new query
  - should give more relevant results now
- Pseudo-Relevance Feedback:
  - just take the top-k documents returned (assuming they are all relevant) and do the same with them
  - can be iterated

BM25

- BM25 = Best Match 25
  - a formula used with great success in the Okapi IR system
  - here is the formula for the weight of some term in some doc.

    $tf^* \cdot \log (N / df)$

    where $tf^* = tf \cdot (k + 1) / (k \cdot (1 - b + b \cdot DL / AVDL) + tf)$

    where tf = term frequeny

    DL = document length

    $AVDL$ = average document length

    $k = 1.5$ and $b = 0.75$ (tuning parameters)
- Outperformed all previous formulas at its time
  - and still one of the best
  - although the theory is more hand-waving than theory

    (which is quite typical for information retrieval research)

BM25 — Derivation

- Replace simple tf by tf* = tf · (k + 1) / (k + tf)
  - is 0 when tf = 0
  - increases as tf increases
  - limit → k + 1 as tf → infinity
- Normalize tf depending on document length
  - alpha = DL / AVDL
  - better: alpha = (1-b) + b · DL / AVDL
  - replace tf → tf / alpha
  - this gives the BM25 formula

Global document score

- Give each document a query-independent score, too
  - very important for web search
  - where some pages are just more important than others
  - Google's PageRank
    - we might do it later in the course
  - simple technical realization:
    - just add a special word IMPORTANCE to every document
    - for each document assign a score to that special word reflectin how important the document is
    - when receiving the query uni freiburg ...
    - ... actually process the query IMPORTANCE uni freiburg
    - this will add the respective score to each document

Precision and Recall

- Consider a particular query
  - Hits = subset of documents returned for that query
  - Relevance of a document = assessed by human
  - Precision = percentage of hits that are relevant
  - Recall = percentage of relevant documents returned as hits
  - Precision @ K = percentage of relevant docs in the top-K
  - Precision at recall 10% (and similarly for other %ages):
    - pick k such that top-k hits contain 10% of all relevant docs
    - Precision @ 10% = percentage of relevant docs in these k docs
  - Average precision :
    - Average of Precision @ 10%, Precision @ 20%, ... (until 100%)

Precision-Recall Graph

- Draw an example:
  - recall levels on x-axis
  - precision at the respective level on y-axis

Precision-Recall Example

- Do an example by hand

F-Measure

- Single value capturing both precision and recall

  $2 \cdot \text{precision} \cdot \text{recall} \ / \ (\text{precision} + \text{recall})$

  – that's just the harmonic mean of the two

  – so it always lies between the two

Relevance

- Some ranking issues have nothing to do with the term weights
  - Chris Buckley
    Why Current Search Engines Fail
    In Proceedings SIGIR 2004
  - http://doi.acm.org/10.1145/1008992.1009132

Top-k query processing

middle		nowhere	
D17	0.5	D23	0.2
D9	0.4	D20	0.1
D21	0.2	D9	0.1
D23	0.2	D17	0.1

- Goal: find the k documents with the highest score sum
  - for k = 1, this is document D17 in the example above

Top-k query processing — Naïve Way

middle		nowhere	
D9	0.2	D9	0.1
D17	0.5	D17	0.1
D21	0.2	D20	0.1
D23	0.2	D23	0.2

1. have each list sorted by doc id

Top-k query processing — Naïve Way

middle		nowhere	
D9	0.2	D9	0.1
D17	0.5	D17	0.1
D21	0.2	D20	0.1
D23	0.2	D23	0.2

1. have each list sorted by doc id

D9	D9	D17	D17	D20	D21	D23	D23
0.2	0.1	0.5	0.1	0.1	0.2	0.2	0.2

2. merge the lists

Top-k query processing — Naïve Way

middle		**nowhere**	
D9	0.2	D9	0.1
D17	0.5	D17	0.1
D21	0.2	D20	0.1
D23	0.2	D23	0.2

1. have each list sorted by doc id

D9	D17	D20	D21	D23
0.3	0.6	0.1	0.2	0.4

2. merge the lists

Top-k query processing — Naïve Way

middle		**nowhere**	
D9	0.2	D9	0.1
D17	0.5	D17	0.1
D21	0.2	D20	0.1
D23	0.2	D23	0.2

1. have each list sorted by doc id

D9	D17	D20	D21	D23
0.3	0.6	0.1	0.2	0.4

2. merge the lists

D17	D23	D9	D21	D20
0.6	0.4	0.3	0.2	0.1

3. sort by score

Top-k query processing

middle		**nowhere**	
D9	0.2	D9	0.1
D17	0.5	D17	0.1
D21	0.2	D20	0.1
D23	0.2	D23	0.2

1. have each list sorted by doc id

D9	D17	D20	D21	D23
0.3	0.6	0.1	0.2	0.4

2. merge the lists

D17	D23	D9	D21	D20
0.6	0.4	0.3	0.2	0.1

3. sort by score

requires full scan of all lists involved

Top-k query processing — More Efficient

middle		nowhere	
D17	0.5	D23	0.2
D9	0.2	D20	0.1
D21	0.2	D9	0.1
D23	0.2	D17	0.1

have each list
sorted by **score**

Top-k query processing — More Efficient

middle		nowhere	
D17	0.5	D23	0.2
D9	0.2	D20	0.1
D21	0.2	D9	0.1
D23	0.2	D17	0.1

have each list
sorted by **score**

D17 [0.5 , 0.7]
D23 [0.2 , 0.7]

all others <= 0.7

Top-k query processing — More Efficient

middle		nowhere	
D17	0.5	D23	0.2
D9	0.2	D20	0.1
D21	0.2	D9	0.1
D23	0.2	D17	0.1

have each list sorted by **score**

D17 [0.5 , 0.6]
D23 [0.2 , 0.4]
D9 [0.2 , 0.3]
D20 [0.1 , 0.3]

all others <= 0.3

Top-k query processing — More Efficient

middle		nowhere	
D17	0.5	D23	0.2
D9	0.2	D20	0.1
D21	0.2	D9	0.1
D23	0.2	D17	0.1

have each list sorted by **score**

D17 [0.5 , 0.6]
D23 [0.2 , 0.4]
D9 [0.2 , 0.3]
D20 [0.1 , 0.3]

all others <= 0.3

we can stop here!

Top-k query processing — Literature

- Celebrated result by Fagin et al :
  - Ronald Fagin and Amnon Lotem and Moni Naor
    Optimal aggregation algorithms for middleware
    Journal of Computer and Systems Sciences 66:614-656, 2003
  - an algorithm with costs that are within a factor of 4m+k of the optimum for each instance
  - but for k = 10 and m = 3, this is already a factor of 22
  - but it can be made to work in practice
    - Bast et al, VLDB 2006
    - sort by score, divide into blocks, then sort blocks by doc id