DAT630
Retrieval Models I.

Search Engines, Chapters 5, 7

04/10/2016

Krisztian Balog | University of Stavanger

So far…

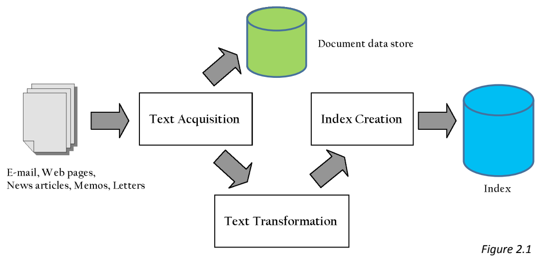


Figure 2.1

Today

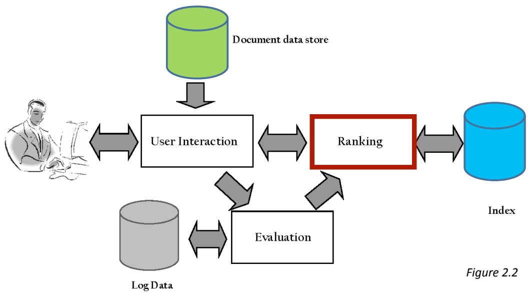


Figure 2.2

Boolean Retrieval

Boolean Retrieval

- Two possible outcomes for query processing
  - TRUE and FALSE (relevance is binary)
  - "Exact-match" retrieval
- Query usually specified using Boolean operators
  - AND, OR, NOT
  - Can be extended with wildcard and proximity operators
- Assumes that all documents in the retrieved set are equally relevant

Boolean Retrieval

- Many search systems you still use are Boolean:
  - Email, library catalog, …
- Very effective in some specific domains
  - E.g., legal search
  - E.g., patent search
  - Expert users

Boolean View of a Collection

- Each row represents the view of a particular term: What documents contain this term?
  - Like an inverted list
- To execute a query
  - Pick out rows corresponding to query terms
  - Apply the logic table of the corresponding Boolean operator

Term	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5	Doc 6	Doc 7	Doc 8
aid	0	0	0	1	0	0	0	1
all	0	1	0	1	0	1	0	0
back	1	0	1	0	0	0	1	0
brown	1	0	1	0	1	0	1	0
come	0	1	0	1	0	1	0	1
dog	0	0	1	0	1	0	0	0
fox	0	0	1	0	1	0	1	0
good	0	1	0	1	0	1	0	1
jump	0	0	1	0	0	0	0	0
lazy	1	0	1	0	1	0	1	0
men	0	1	0	1	0	0	0	1
now	0	1	0	0	0	1	0	1
over	1	0	1	0	1	0	1	1
party	0	0	0	0	0	1	0	1
quick	1	0	1	0	0	0	0	0
their	1	0	0	0	1	0	1	0
time	0	1	0	1	0	1	0	0

Example Queries

Term	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5	Doc 6	Doc 7	Doc 8
dog	0	0	1	0	1	0	0	0
fox	0	0	1	0	1	0	1	0

dog ∧ fox	?
dog ∨ fox	?
dog ¬ fox	?
fox ¬ dog	?

dog AND fox → ?

dog OR fox → ?

dog AND NOT fox → ?

fox AND NOT dog → ?

Example Query

good AND party AND NOT over

Term	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5	Doc 6	Doc 7	Doc 8
good	0	1	0	1	0	1	0	1
party	0	0	0	0	0	1	0	1
over	1	0	1	0	1	0	1	1

g ∧ p	0	0	0	0	0	1	0	1
over	1	0	1	0	1	0	1	1

good AND party → Doc 6, Doc 8

g ∧ p ¬ o	0	0	0	0	0	1	0	0

good AND party AND NOT over → Doc 6

Example of Query (Re)formulation

lincoln

- Retrieves a large number of documents
- User may attempt to narrow the scope

president AND lincoln

- Also retrieves documents about the management of the Ford Motor Company and Lincoln cars

Ford Motor Company today announced that Darrly Hazel will succeed Brian Kelly as **president** of **Lincoln** Mercury.

Example of Query (Re)formulation

- User may try to eliminate documents about cars

president AND lincoln
AND NOT (automobile OR car)

- This would remove any document that contains even of the single mention of "automobile" or "car"
- For example, sentence in biography

Lincoln's body departs Washington in a nine-**car** funeral train.

Example of Query (Re)formulation

- If the retrieved set is too large, the user may try to further narrow the query by adding additional words that occur in biographies

president AND lincoln
AND (biography OR life OR birthplace)
AND NOT (automobile OR car)

- This query may do a reasonable job at retrieving a set containing some relevant documents
- But it does not provide a **ranking** of documents

Example

- WestLaw.com: Largest commercial (paying subscribers) legal search service
- Example query:
  - *What is the statute of limitations in cases involving the federal tort claims act?*
  - **LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM**
    - ! = wildcard, /3 = within 3 words, /S = in same sentence

Boolean Retrieval

- Advantages
  - Results are relatively easy to explain
  - Many different features can be incorporated
  - Efficient processing since many documents can be eliminated from search
  - We do not miss any relevant document

Boolean Retrieval

- Disadvantages
  - Effectiveness depends entirely on user
    - Simple queries usually don't work well
    - Complex queries are difficult to create accurately
  - No ranking
  - No control over result set size: either too many docs or none
  - What about partial matches? Documents that "don't quite match" the query may be useful also

Ranked Retrieval

General Scoring Formula

$$score(d,q) = \sum_{t \in q} w_{t,d} \cdot w_{t,q}$$

Relevance score
It is computed for each document *d* in the collection for a given input query *q*

Documents are returned in decreasing order of this score

It is enough to consider terms in the query

Term's weight in the document

Term's weight in the query

Example 1: Term presence/absence

- The score is the number of matching query terms in the document

$$score(d,q) = \sum_{t \in q} w_{t,d} \cdot w_{t,q}$$

$$w_{t,q} = f_{t,q}$$

$$w_{t,d} = \begin{cases} 1, & f_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

- $f_{t,d}$ is the number of occurrences of term *t* in document *d*
- $f_{t,q}$ is the number of occurrences of term *t* in query *q*

Term Weighting

- Instead of using raw term frequencies, assign a weight that reflects the term's importance

Example 2: Log-frequency Weighting

$$w_{t,d} = \begin{cases} 1 + \log f_{t,d}, & f_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

Raw term frequency →

$f_{t,d}$	$w_{t,d}$
0	0
1	1
2	1.3
10	2
1000	4

Example 2: Log-frequency Weighting

$$score(d,q) = \sum_{t \in q} w_{t,d} \cdot w_{t,q}$$

$$score(d,q) = \sum_{t \in q} (1 + \log f_{t,d}) \cdot f_{t,q}$$

Query Processing

- Strategies for processing the data in the index for producing query results
- Document-at-a-time
  - Calculates complete scores for documents by processing all term lists, one document at a time
- Term-at-a-time
  - Accumulates scores for documents by processing term lists one at a time
- Both approaches have optimization techniques that significantly reduce time required to generate scores

Document-at-a-Time

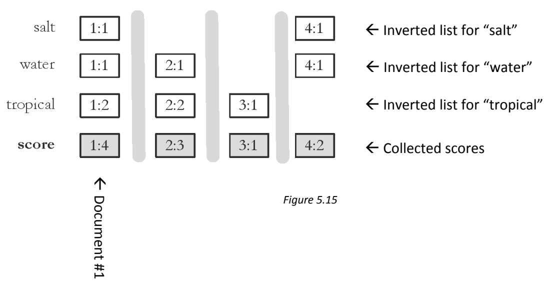


Figure 5.15

Term-at-a-Time

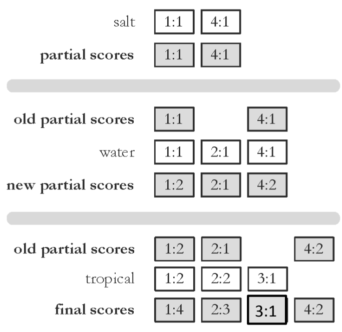


Figure 5.17

The Vector Space Model

The Vector Space Model

- Basis of most IR research in the 1960s and 70s
- Still used
- Provides a simple and intuitively appealing framework for implementing
  - Term weighting
  - Ranking
  - Relevance feedback

Representation

- Documents and query represented by a **vector of term weights**

$$D_i = (d_{i1}, d_{i2}, \ldots, d_{it}) \qquad Q = (q_1, q_2, \ldots, q_t)$$

- Collection represented by a matrix of term weights

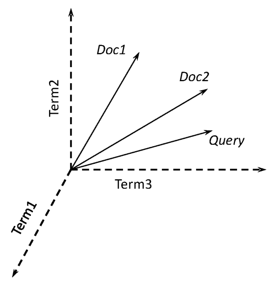
	$Term_1$	$Term_2$	$\ldots$	$Term_t$
Doc_1	d_{11}	d_{12}	$\ldots$	d_{1t}
Doc_2	d_{21}	d_{22}	$\ldots$	d_{2t}
$\vdots$	$\vdots$			
Doc_n	d_{n1}	d_{n2}	$\ldots$	d_{nt}

Bag of Words Model

- Vector representation doesn't consider the ordering of words in a document
- "*John is quicker than Mary*" and "*Mary is quicker than John*" have the same vectors

Scoring Documents

- Documents "near" the query's vector (i.e., more similar to the query) are more likely to be relevant to the query

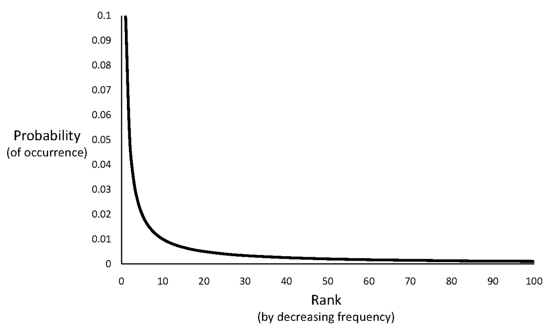


Scoring Documents

- The score for a document is computed using the cosine similarity of the document and query vectors

$$cosine(d, q) = \frac{\sum_t w_{t,d} \cdot w_{t,q}}{\sqrt{\sum_t w_{t,d}^2}\sqrt{\sum_t w_{t,q}^2}}$$

Zipf's Law



Weighting Terms

- Intuition
  - Terms that appear often in a document should get high weights
    - The more often a document contains the term "dog", the more likely that the document is "about" dogs
  - Terms that appear in many documents should get low weights
    - E.g., stopword-like words
- How do we capture this mathematically?
  - Term frequency
  - Inverse document frequency

Term Frequency (TF)

- Reflects the importance of a term in a document (or query)
- Variants
  - binary $tf_{t,d} = \{0, 1\}$
  - raw frequency $tf_{t,d} = f_{t,d}$
  - normalized $tf_{t,d} = f_{t,d}/|d|$
  - log-normalized $tf_{t,d} = 1 + \log f_{t,d}$
  - …
- $f_{t,d}$ is the number of occurrences of term k in the document and |d| is the length of d

Inverse Document Frequency (IDF)

- Reflects the importance of the term in the collection of documents
  - The more documents that a term occurs in, the less *discriminating* the term is between documents, consequently, the less useful for retrieval

$$idf_t = \log \frac{N}{n_t}$$

  - where N is the total number of document and n_t is the number of documents that contain term t
  - log is used to "dampen" the effect of IDF

Term Weights

- Combine TF and IDF weights by multiplying them:

$$tfidf_{t,d} = tf_{t,d} \cdot idf_t$$

  - Term frequency weight measures importance in document
  - Inverse document frequency measures importance in collection

Scoring Documents

- The score for a document is computed using the cosine similarity of the document and query vectors

$$cosine(d,q) = \frac{\sum_t w_{t,d} \cdot w_{t,q}}{\sqrt{\sum_t w_{t,d}^2}\sqrt{\sum_t w_{t,q}^2}}$$

$$cosine(d,q) = \frac{\sum_t tfidf_{t,d} \cdot tfidf_{t,q}}{\sqrt{\sum_t tfidf_{t,d}^2 \sum_t tfidf_{t,q}^2}}$$

Scoring Documents

- It also fits within our general scoring scheme:
  - Note that we only consider terms that are present in the query

$$Score(q,d) = \sum_{t \in q} w_{t,q} \cdot w_{t,d}$$

$$w_{t,q} = \frac{tfidf_{t,q}}{\sqrt{\sum_t tfidf_{t,q}^2}} \qquad w_{t,d} = \frac{tfidf_{t,d}}{\sqrt{\sum_t tfidf_{t,d}^2}}$$

Variations on Term Weighting

- It is possible to use different term weighting for documents and for queries, for example:

weighting scheme	document term weight	query term weight
1	$f_{i,j} * \log \frac{N}{n_i}$	$(0.5 + 0.5 \frac{f_{i,q}}{max_i \ f_{i,q}}) * \log \frac{N}{n_i}$
2	$1 + \log f_{i,j}$	$\log(1 + \frac{N}{n_i})$
3	$(1 + \log f_{i,j}) * \log \frac{N}{n_i}$	$(1 + \log f_{i,q}) * \log \frac{N}{n_i}$

- See also: https://en.wikipedia.org/wiki/Tf-idf for further variants

Difference from Boolean Retrieval

- Similarity calculation has two factors that distinguish it from Boolean retrieval
  - Number of matching terms affects similarity
  - Weight of matching terms affects similarity
- Documents can be *ranked* by their similarity scores

Exercise

BM25

BM25

- BM25 was created as the result of a series of experiments
- Popular and effective ranking algorithm
- The reasoning behind BM25 is that good term weighting is based on three principles
  - Inverse document frequency
  - Term frequency
  - Document length normalization

BM25 Scoring

$$score(d,q) = \sum_{t \in q} \frac{f_{t,d} \cdot (1+k_1)}{f_{t,d} + k_1(1 - b + b\frac{|d|}{avgdl})} \cdot idf_t$$

- Parameters
  - k_1: calibrating term frequency scaling
  - b: document length normalization
- Note: several slight variations of BM25 exist!

BM25: An Intuitive View

$$score(d,q) = \sum_{\boxed{t \in q}} \frac{f_{t,d} \cdot (1+k_1)}{f_{t,d} + k_1(1 - b + b\frac{|d|}{avgdl})} \cdot idf_t$$

Terms common between the document and the query => good

BM25: An Intuitive View

$$score(d,q) = \sum_{t \in q} \frac{\boxed{f_{t,d}} \cdot (1+k_1)}{f_{t,d} + k_1(1 - b + b\frac{|d|}{avgdl})} \cdot idf_t$$

Repetitions of query terms in the document => good

BM25: An Intuitive View

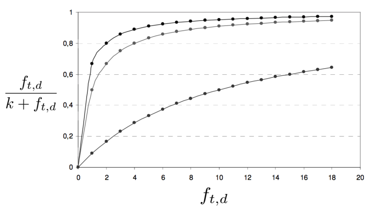
$$score(d,q) = \sum_{t \in q} \boxed{\frac{f_{t,d} \cdot (1+k_1)}{f_{t,d} + k_1(1 - b + b\frac{|d|}{avgdl})}} \cdot idf_t$$

Term saturation: repetition is less important after a while

BM25: An Intuitive View

$$score(d,q) = \sum_{t \in q} \boxed{\frac{f_{t,d} \cdot (1+k_1)}{f_{t,d} + k_1(1 - b + b\frac{|d|}{avgdl})}} \cdot idf_t$$

Term saturation



$\frac{f_{t,d}}{k+f_{t,d}}$ for some k > 0

Asymptotically approaches 1

Middle line is *k=1*
Upper line is lower *k*
Lower line is higher *k*

BM25: An Intuitive View

$$score(d,q) = \sum_{t \in q} \frac{f_{t,d} \cdot (1+k_1)}{f_{t,d} + k_1\boxed{(1 - b + b\frac{|d|}{avgdl})}} \cdot idf_t$$

Soft document normalization taking into account document length
Document is more important if relatively long (w.r.t. average)

BM25: An Intuitive View

$$score(d,q) = \sum_{t \in q} \frac{f_{t,d} \cdot (1 + k_1)}{f_{t,d} + k_1(1 - b + b\frac{|d|}{avgdl})} \cdot \boxed{idf_t}$$

Common terms less important

Parameter Setting

- k_1: calibrating term frequency scaling
  - 0 corresponds to a binary model
  - large values correspond to using raw term frequencies
  - k_1 is set between 1.2 and 2.0, a typical value is 1.2
- b: document length normalization
  - 0: no normalization at all
  - 1: full length normalization
  - typical value: 0.75