The University of Manchester

COMP34711
Week 3

Part 2
Querying and ranking

Goran Nenadic

with examples from the IIR book

Outline

- The Boolean model
- Ranked retrieval
  - Vector space model for IR
  - tf*idf
  - Ranking documents
- Measuring the quality of IR
  - Standard metrics
- Enhancing IR
  - User behaviour

Boolean queries

- The simplest query model: find all documents from the collection that fully match the query
  - Binary outcome for each document: yes/no
- Use operators
  - AND (set intersection)
  - OR (set union)
  - NOT (set difference, complement)
- E.g.

drug AND **approach** = all documents that contain both **query words**

drug OR **approach** = all documents that either of the **query words**

drug AND NOT **approach** = all documents that contain **drug** but do not contain **approach**

Many search systems 'hide' Boolean functionality (e.g. space is used as AND)

Boolean query processing: AND

- Consider processing the query:

  Brutus *AND* ***Caesar***

  1. Locate ***Brutus*** in the Dictionary
     - Retrieve its postings
  2. Locate ***Caesar*** in the Dictionary
     - Retrieve its postings

Brutus ⇒ 2 → 4 → 8 → 16 → 32 → 64 → 128

Caesar ⇒ 1 → 2 → 3 → 5 → 8 → 13 → 21 → 34

  3. "AND-merge" the two postings:

Boolean query processing: AND

- "AND-merge":
  - walk through the two postings simultaneously (moving/ progressing through the postings with lower docID) until there is no possible matches
  - <u>crucial</u>: postings are sorted by docID.

Brutus 2 → 4 → 8 → 16 → 32 → 64 → 128

Caesar 1 → 2 → 3 → 5 → 8 → 13 → 21 → 34

2 → 8

(IIR book)

The University of Manchester

Boolean model: **pros**

- Simple model: everything is considered as a set of terms
- Easy/efficient to implement
- Precise
  - Document either matches or does not match
- Widely used for commercial, legal retrieval and for specialist searches
  - Long, precise queries
- Works well when we know what we want: user feels in control

Boolean model: **cons**

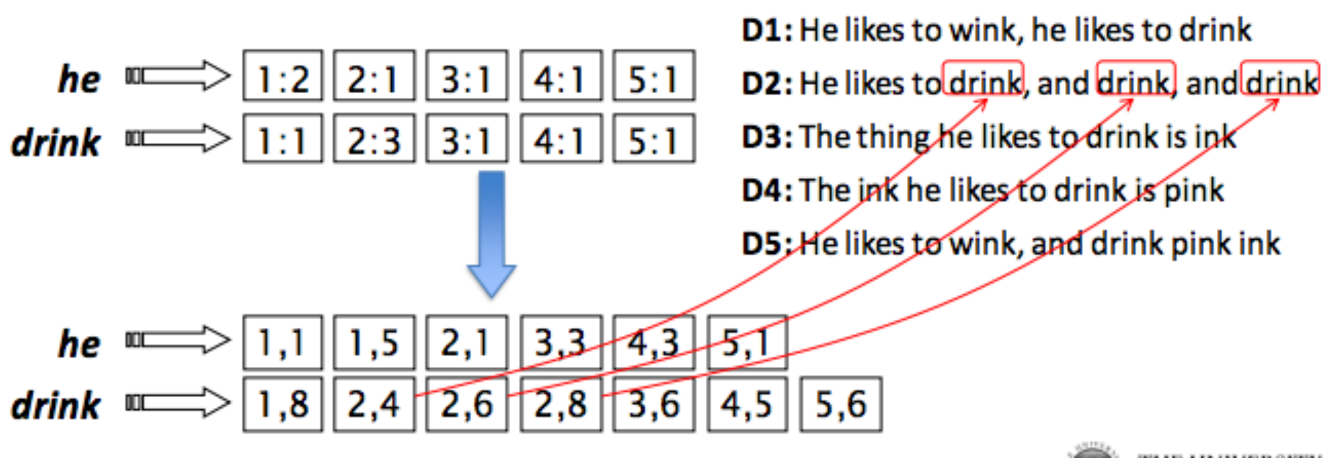
- Users are not good in formulating queries: possible confusion with natural language
  - cats OR dogs
  - cats OR dogs AND NOT horses
- "feast or famine"
  - AND gives too few or no results
    (the more ANDs, the smaller the result set)
  - OR gives too many
- Basic Boolean expressions too limiting for information needs?

Boolean model: **cons**

- No relevance ordering/ranking of results
  - In principle, all retrieved documents are good
    - But note: the BoW model does not use word order in documents
  - How to ‘read’ them if there are too many?
  - Some notion of additional relevance could be added:
    - date reverse order of document creation?
    - the frequency of query terms in matched documents?
    - proximity of query terms in documents?

Extended Boolean model

- **Proximity operators**
  - Embed term positions to the inverted index (proximity index)



  - Queries can then refer to these, e.g.
    - /n (e.g. /3 – within 3 words)
    - /s = in same sentence, /p in same paragraph
    - +s = term1 must precede term2 in same sentence

Ranked retrieval

- In Boolean query model
  - Documents match or don't match
  - Results are unranked

- Documents that are 'close' are not retrieved

  social AND worker AND union
  - Will not retrieve a document that has social and worker in it if union is not not mentioned, although it may mention UCU or TUC
  - Boolean querying is too rigid

Ranked retrieval

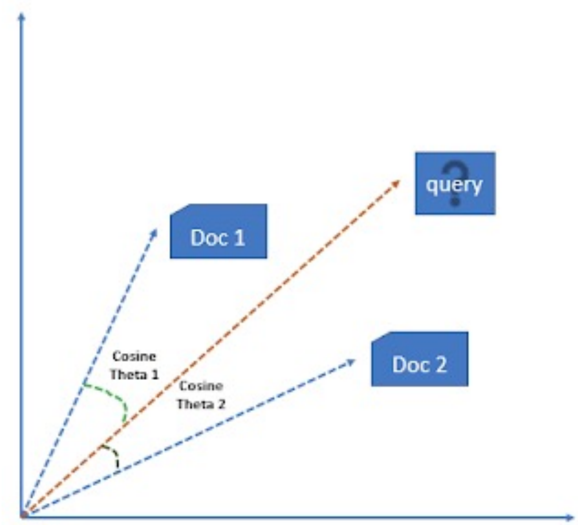
- Introducing *similarity* and *ranking* of matched documents
  - Attempt more than exact matching queries with docs
  - Score each document to say *how well it matches* query
    - E.g. assign a real number score in range 0..1
  - Typically aim to get top *K* (10?) ones correct
    - As user will likely not look much further

- Idea: use vector representation for both documents and queries, and calculate their similarity

Vector representations

- Represent both documents and queries as vectors in the same space

- Then, rank (all) documents according to their <u>proximity</u> to the given query in that space
  - Rank more relevant documents higher than less relevant documents
  - Recall: We do this because we want to get away from the “you’re-either-in-or-out” Boolean model.

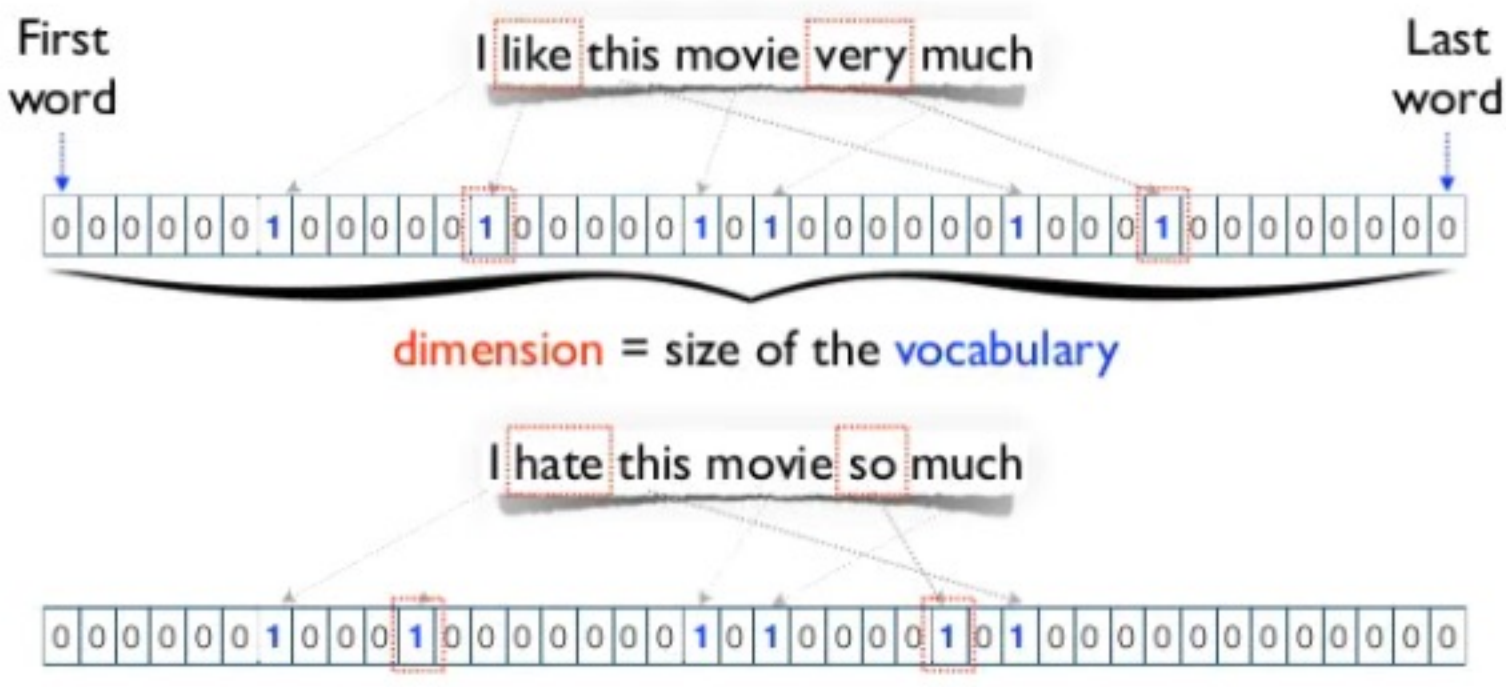


(IIR book)

The University of Manchester

Weights

- What weight to use to represent terms that appear in documents/queries?



Term frequency (tf)

- Term frequency $tf_{t,d}$ of term t in document d is defined as the number of times that t occurs in d.
- How to use tf when computing query-document match scores?
- Raw term frequency is not what we want:
  - Document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term
  - *But not 10 times more relevant*
- Relevance does not increase proportionally with term frequency

NB: frequency = count in IR

(IIR book)

Document frequency (df)

- Document frequency = in how many documents term appears
- Consider term in query that is rare in collection (e.g., *arachnocentric*)
- A document containing this term is very likely to be relevant to a query that contains *arachnocentric*

  → We want a high weight for rare terms (like *arachnocentric*)

- Rare terms are more informative and discriminative than frequent terms for IR

(IIR book)

Document frequency (df)

- Consider a query term that is frequent in collection (e.g., *high*, *increase*, *line*)

- A document containing such a term is more likely to be relevant than a document that doesn't

- But it's not a sure indicator of relevance

  → For frequent terms, we want high positive weights, but lower weights than for rare terms

(IIR book)

Collection vs. document frequency

- The collection frequency of *t* is the number of occurrences of *t* in the collection, counting multiple occurrences (within the same document)
- Example:

Word	Collection frequency	Document frequency
insurance	10440	3997
try	10422	8760

  – Which word is a "better" search term (and should get a higher weight)?

(IIR book)

Inverse document frequency

- Document frequency (df_t) is the number of documents that contain term t
  - df_t is an inverse measure of the "informativeness" of t
  - $\mathrm{df}_t \leq N$

- We define the idf (inverse document frequency) of t by

$$\mathrm{idf}_t = \log_{10} (N/\mathrm{df}_t)$$

  - We use $\log(N/\mathrm{df}_t)$ instead of N/df_t to "dampen" the effect of idf
  - The base of the log is immaterial.

(IIR book)

idf example

Collection of N = 1,000,000 documents

term	df_t	idf_t	
Calpurnia	1	6	(= log(1M/1))
animal	100	4	(= log(1M/100))
sunday	1,000	3	(= log(1000))
fly	10,000	2	(= log (100))
under	100,000	1	(= log(10))
the	1,000,000	0	(= log(1))

- There is one idf value for each term *t* in a collection

$$\mathrm{idf}_t = \log_{10} (N/\mathrm{df}_t)$$

$$\mathrm{idf}_{fly} = \log_{10} (1{,}000{,}000/10{,}000) = \log_{10} 100 = 2$$

(IIR book)

Activity – see BB

tf.idf weighting

- The tf-idf weight of a term is the product of its tf weight and its idf weight

$$\mathrm{tf.idf}_{t,d} = (1 + \log_{10} \mathrm{tf}_{t,d}) \times \log_{10}(N / \mathrm{df}_t)$$

- Best known weighting scheme in information retrieval
  – Note alternative notations: tf-idf, tf x idf, tf*idf, tfidf
- Increases
  – with the number of occurrences within a document
  – with the rarity of the term in the collection

(IIR book)

MANCHESTER
1824
The University of Manchester

frequency

Term-document count matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Each document is a count vector in $\mathbb{N}^v$

(IIR book)

Binary → count → tf*idf weight matrix

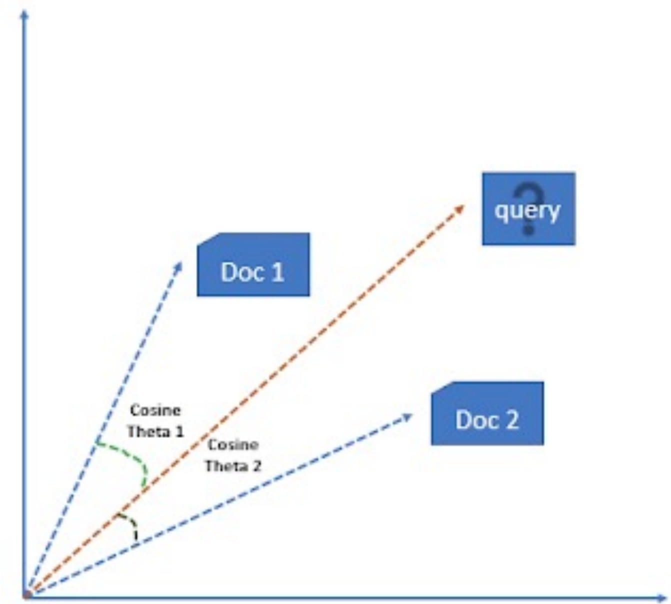
	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

Each document is now represented by a real-valued vector of tf*idf weights $\in R^{|V|}$

(IIR book)

Vector Space Model for IR

- Documents and Queries are presented as vectors
- Match(Q,D) = distance between vectors
- Which distance to use?
  - Euclidean Distance?
    - Distance between the endpoints of the two vectors
    - Large for vectors of diff. lengths
  - Angle between the document and the query
    - Use cosine of the angle

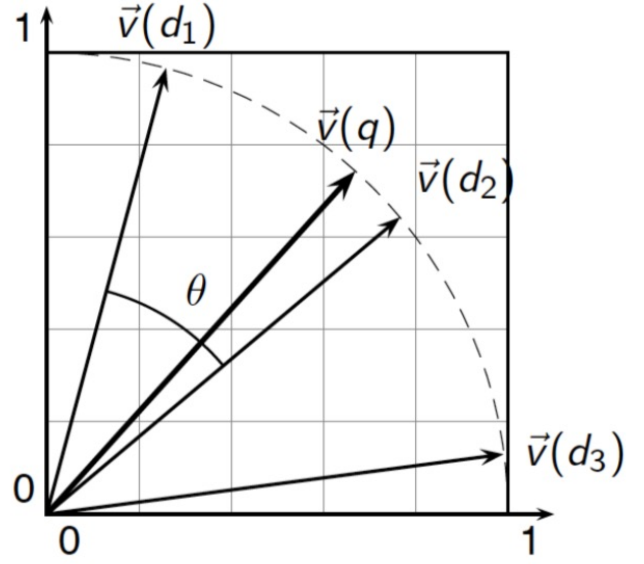


(IIR book)

The University of Manchester

Cosine distance

- Need to normalise for length to ensure fair comparison
  - Long and short documents then have comparable weights
- Dividing a vector by its ***norm*** makes it a unit (length) vector (on surface of unit hypersphere)



(IIR book)

Cosine distance

- For ***length-normalized vectors***, cosine similarity is the dot (or scalar) product:

$$\cos(\vec{q},\vec{d}) = \vec{q} \bullet \vec{d} = \sum_{i=1}^{|V|} q_i d_i$$

- Easy and efficient to calculate
  - Note that we need only the values for terms that appear in both the document and query.
- Use cosine values to rank the document based on their similarity (i.e. distance) to the query

(IIR book)

Document ranking for a query

$$\text{Score}(q,d) = \sum_{t \in q \cap d} \text{tf.idf}_{t,d}$$

- Score of a document *d* is sum over all tf.idf weights of each query term *t* found in *d*
- Indexing: for each term and document calculate tf*idf$_{t,d}$
- Typical output: a list of documents ranked according to score (q,d):

```
1 0 710 0 0.9234 0
1 0 213 0 0.7678 0
1 0 103 0 0.6761 0
1 0 13  0 0.6556 0
1 0 501 0 0.4301 0
```

Query id

document id

score

Return top K documents

Summary of the steps

- Pre-process each document
  - tokenisation, stop-words removal (plus maybe some normalization – stemming, spelling corrections)
  - decide what will be index terms and calculate their tf.idf
- Represent each document as a weighted tf.idf vector
- Represent the query as a weighted tf.idf vector
- Compute the (cosine) similarity score for the query vector and each document vector
- Rank documents with respect to the query by score
- Return the top K to the user

(IIR book)