Introduction to Information Retrieval

Prof Fabio Crestani
Faculty of Informatics
University of Lugano (USI)
Switzerland

PROMISE
Winter School 2013
Bressanone, Italy

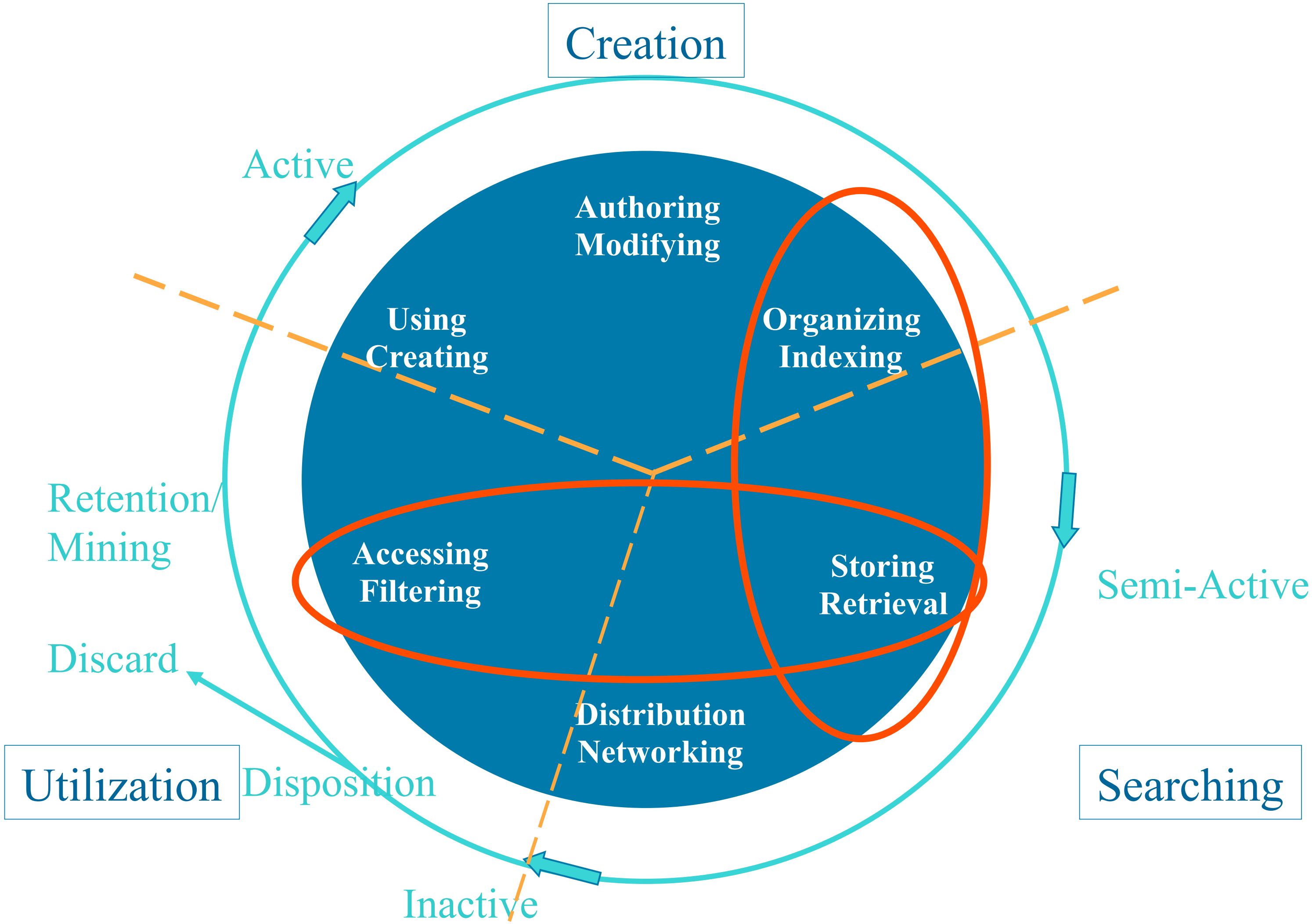
Aim of this lecture

- Review for you some of the main concepts of Information Retrieval (IR)
- Provide an understanding of the architecture and functional specification on an IR systems
- Prepare you to better understand more advanced concepts to follow

Outline

- What is Information Retrieval (IR)?
  - How is it different from Databases?
  - Why is it hard?
- Functional specification of an IR system
- The IR process
  - Indexing
  - Retrieval
- Final considerations

Information Life Cycle



Key issues

- How to describe information resources or information-bearing objects in ways that they can be effectively used by those who need to use them
  - Organizing / Indexing / Storing
- How to find the appropriate information resources or information-bearing objects for someone's (or your own) needs
  - Retrieving / Accessing / Filtering

Information access systems

- The effective use of information requires efficient and effective access to it
- There are several technologies for information access
  - *Database*
  - *Information Retrieval*
  - Digital Libraries
  - Information Filtering
  - Categorisation
  - Expert systems
- Different technologies are needed because information can be in different forms and for different uses

Information is different from data

Types (structured inf.)

Name	Age	Salary	Date joined
String	**Int**	**Int**	**Date**
Donald	25	£50 000	1/3/01
Mickey	52	£100 000	1/2/99

- "Jules Verne wrote *20,000 Leagues Under The Sea* and *Around The World In 80 Days*. He died in 1905."

Words (unstructured inf.)

And so are search statements

- SQL
  - SELECT Name FROM Employee WHERE Age BETWEEN 30 AND 40
  - Artificial language
  - Complete description
  - Exact description

Google
Top 10 Gaining Queries

Week of Jan. 14, 2013

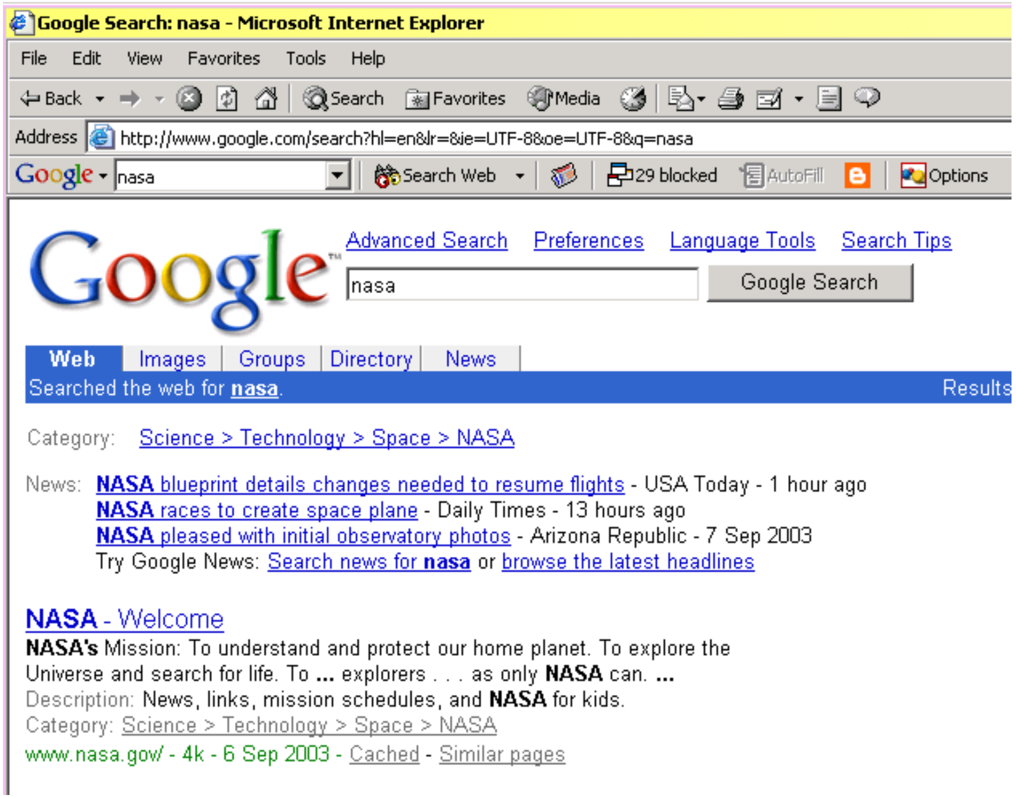
1. chatroulette
2. ipad
3. justin bieber
4. nicki minaj
5. friv
6. myxer
7. katy perry
8. twitter
9. Gamezer
10. facebook

And so is what we get back

–SELECT Name FROM Employee WHERE Age BETWEEN 30 AND 40

gives Names

-known result type (list of Names)



Relevance

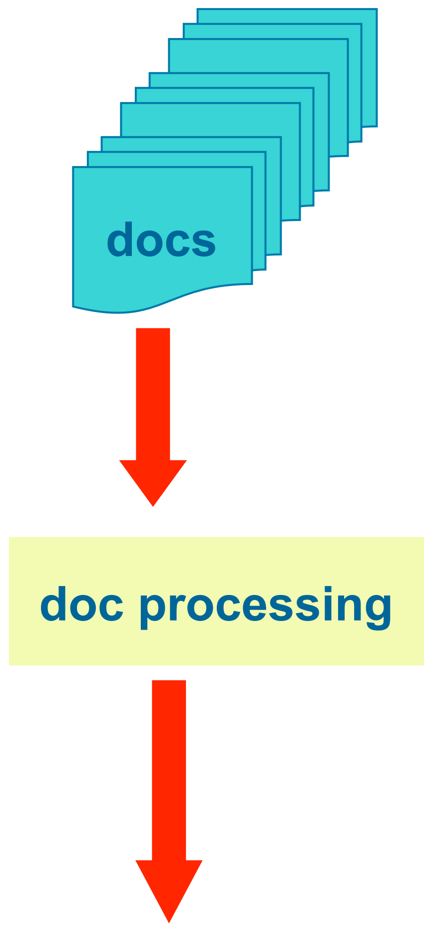
- Relevance is the core concept in IR, but nobody has a good definition
  - Relevance = useful
  - Relevance = topically related
  - Relevance = new
  - Relevance = interesting
  - Relevance = ???
- However we still want *relevant information*

Types of search systems

	Structured	Unstructured
Data	Typed	Untyped
Model	Deterministic	Prob./Sim.
Matching	Exact	Partial
Query specification	Complete	Incomplete
Query language	Artificial	Natural
Items wanted	Matching	Relevant
Error sensitivity	High	Low

Why is this hard?

- Documents/images/video/speech/etc are complex
- We need some representation but
  - Semantics
    - What words mean
  - Natural language
    - How we say things
- Computers cannot deal with these easily



Why is IR hard?

- Context

Sponge

Sponge Bob

- Opinion

Funny

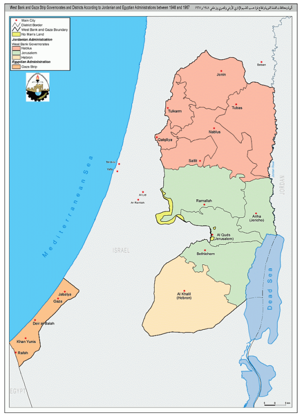
Talented

Honest

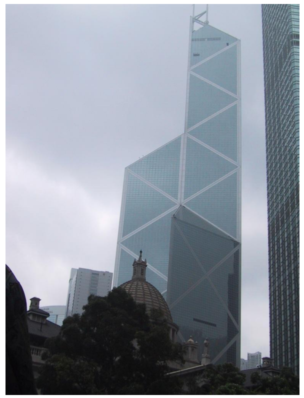
Why is IR hard?

- Semantics

Bank note



West **bank**



Bank of China

Why is this hard?

- Information needs must be expressed as a query
  - But users don't often know what they want
- Problems
  - Verbalising information needs
  - Understanding query syntax
  - Understanding search engines

Information need

query

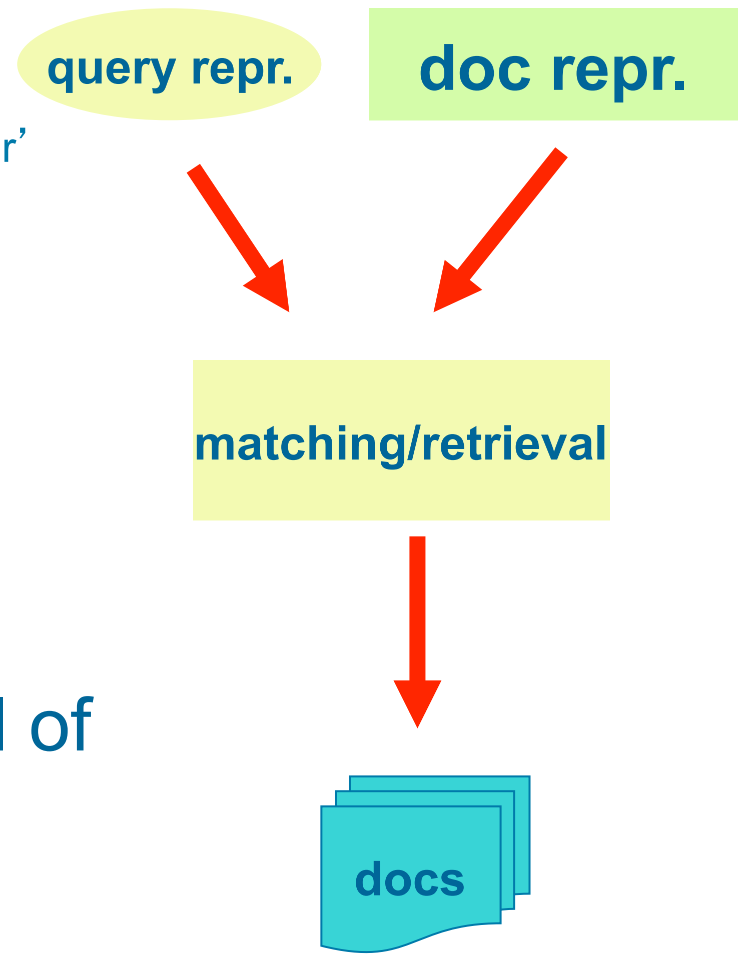
Search

vs

Guess what I mean

Why is this hard?

- Queries are
  - under-specified
    - ‘uefa’ ‘brad pitt’ ‘big brother’
  - ambiguous
    - ‘jordan’
  - context-sensitive
  - represent different types of search
    - E.g. decision making
    - background search
    - fact search
- IR means dealing with all of this



Why is this hard?

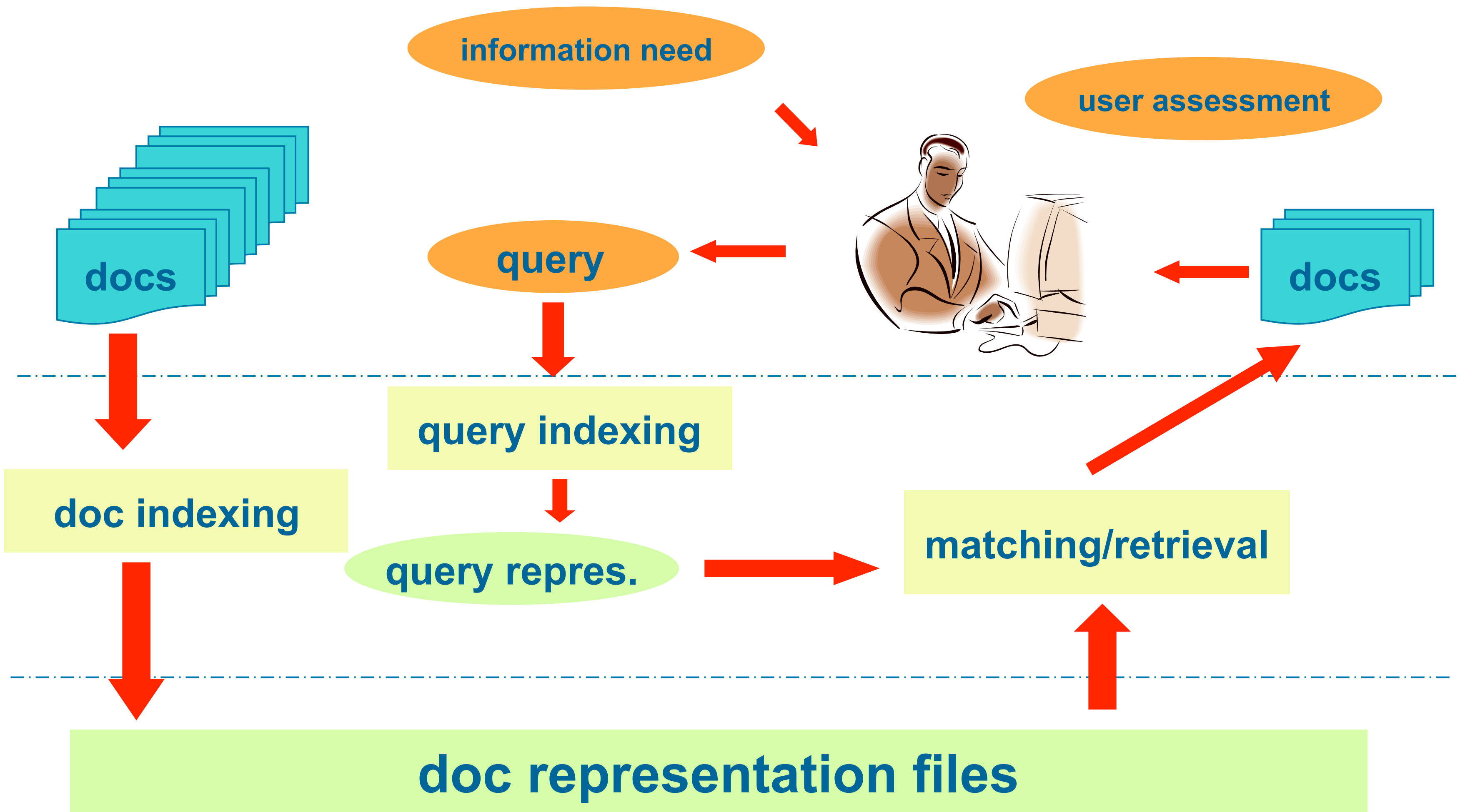
- Scale is also an issue
  - 357+ terabytes of *print* information produced a year (terabyte = 1000 gigabytes)
  - Plus tv stations (video), radio stations (speech), specialist data (satellite image, medical images, music, etc…)
  - Estimate > 600 petabytes per year
    - (1 pb = 1000 terabytes)
    - Of this we can perhaps access about 100pb
- IR means fast and scalable solutions

Why is this hard?

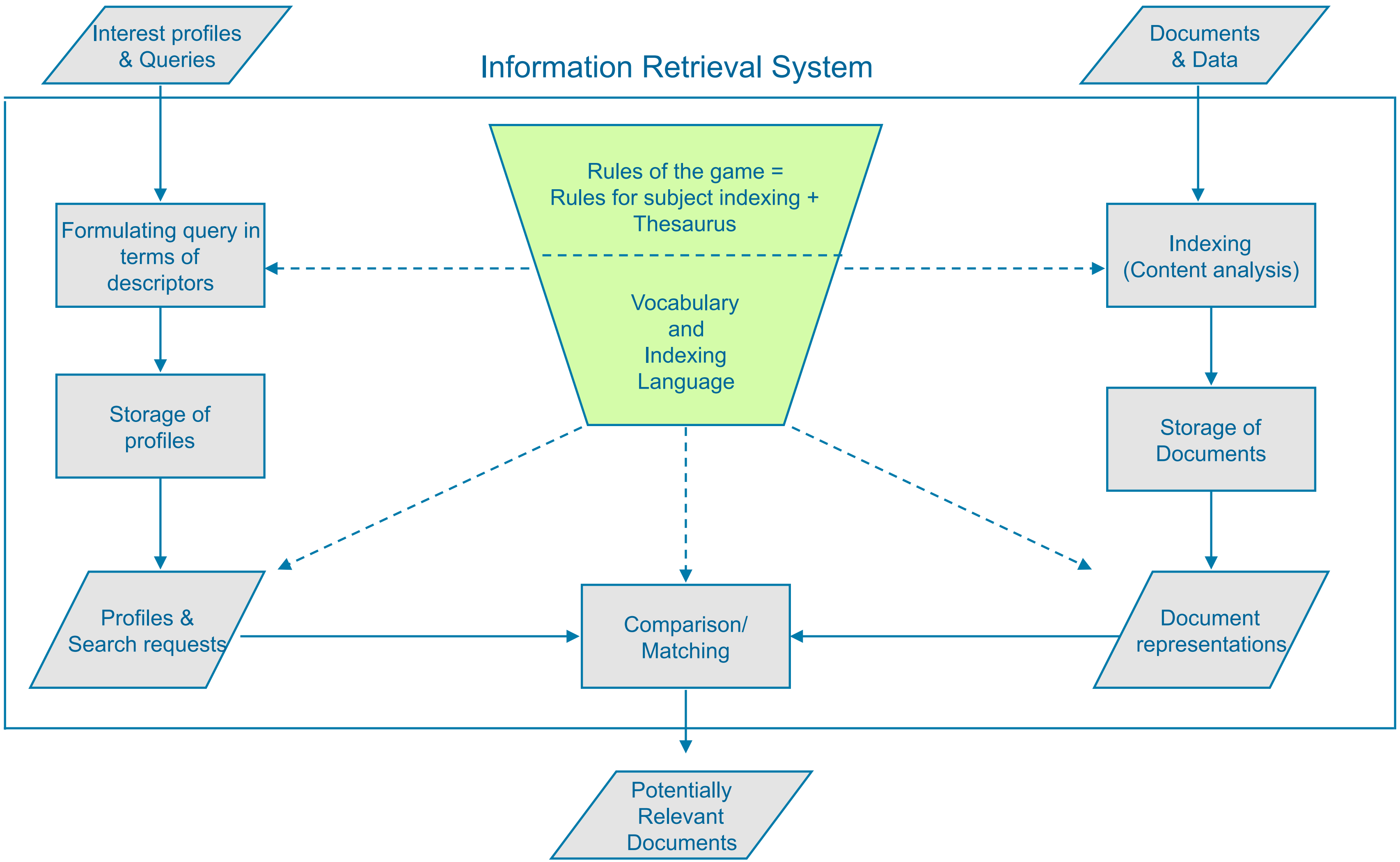
- Information is often dynamic
  - News
  - Web pages
  - Weather maps
  - Etc
- And so are queries
  - Searchers may change information need whilst searching
- IR must cope with change in data and searcher

Questions?

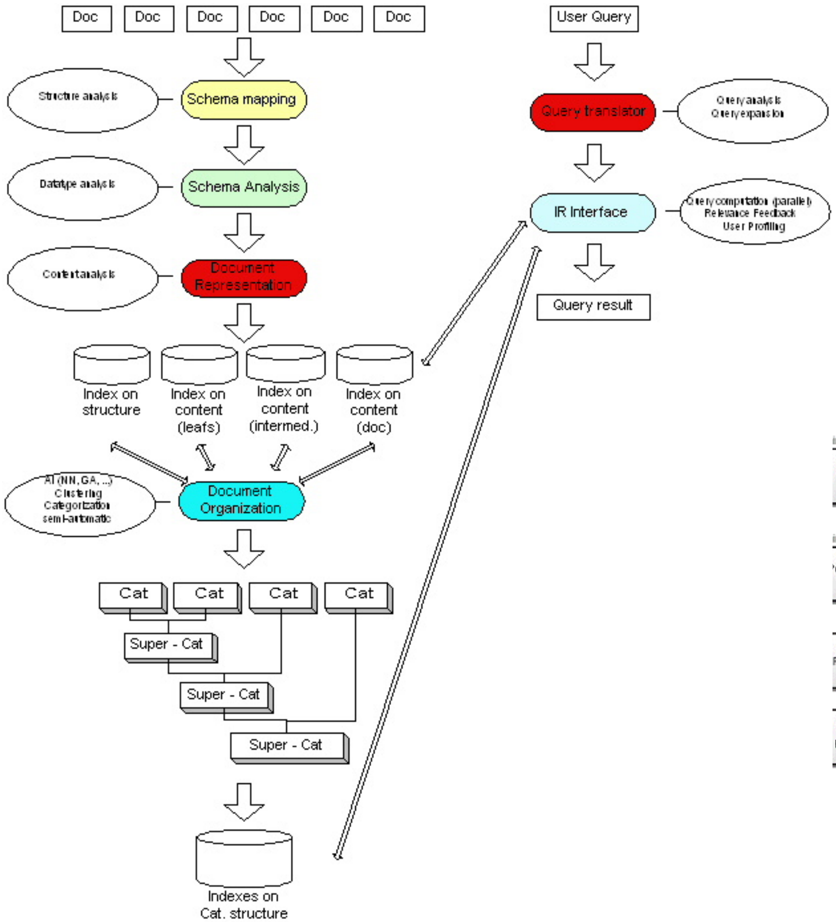
The IR process

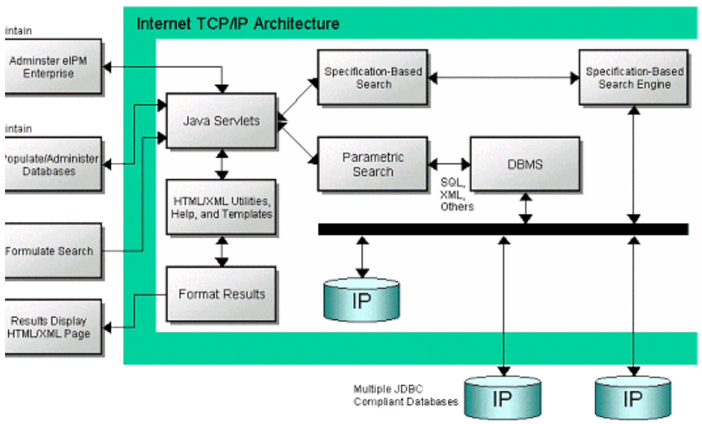


Functional specs of an IR System



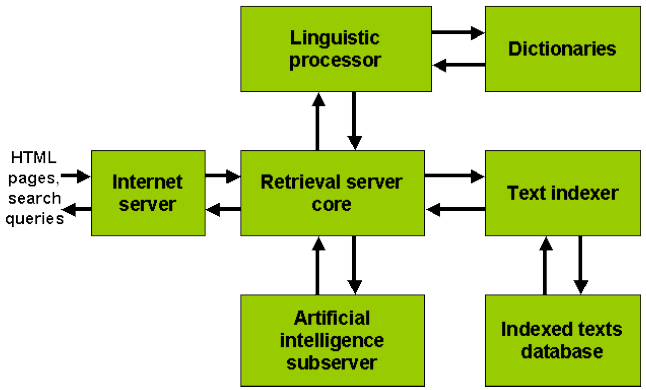
Architecture of an IR system: examples

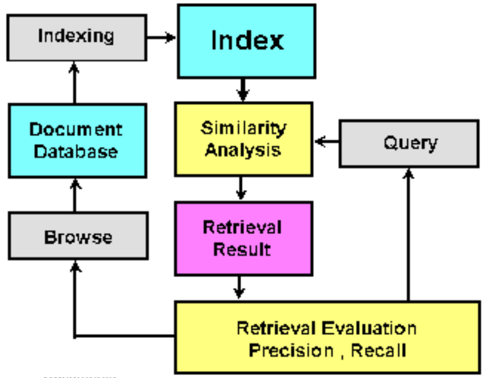


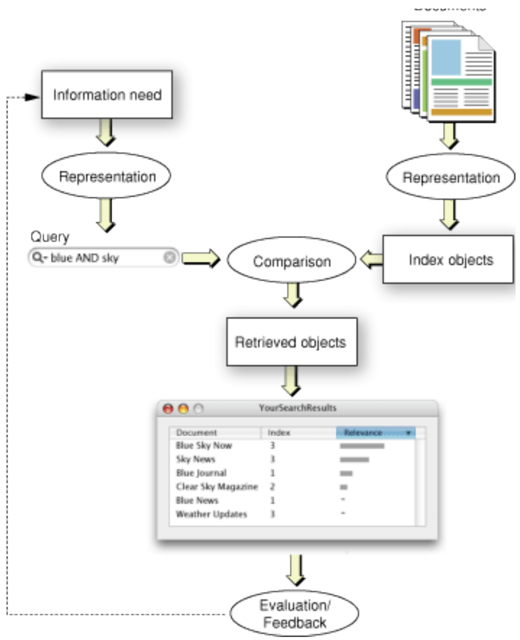


Architecture of an IR system: examples

STRUCTURAL SCHEME OF THE INTELLIGENT INFORMATION RETRIEVAL ENGINE







IR processes

- Main IR processes:
  - Document indexing
  - Query indexing
  - Document retrieval
  - Results visualisation
  - Relevance feedback
- We will study these processes in detail in the next few slides

Questions?

Indexing: outline

- How do we process text for content analysis?
  - Different steps of the indexing process
- Statistical properties of text
  - Zipf distribution
- Term weighting
  - Most important term weighting functions

Indexing process

- Indexing task: “what is this document about?”
- Indexing is carried out in a number of steps:
  1. Character encoding
  2. Language recognition
  3. Segmentation and tokenisation
  4. Phrase identification and named entity recognition
  5. Term normalisation
  6. Stop word removal
  7. Feature normalisation (stemming)
  8. Term weighting

1. Character encoding

- Character encoding is a binary representation of the native language alphabet
  - Usually one-byte (ASCII), but some languages require double-byte encoding (e.g. Japanese, Chinese)
- UNICODE standard for representation of all world's languages
- Problem:
  - Support native codes or transform to UNICODE for processing and retrieval?

Native language encoding

- Language (alphabet) specific native encoding standards
  - Chinese: GB, Big5
  - Western Europe: ISO-8859-1 (Latin1)
  - Russian: KOI-8, ISO-8859-5, CP-1251
- Problems:
  - Not every system follows a standard
  - Need to know which standard is being used
    - In HTML we have the "Content Type" header field
  - Not all standards are comparable
  - Difficult to move from one standard to another

UNICODE / ISO 10646

- Single 16-bit (2-byte) encoding designed to encompass all world's languages
  - 16 bits = 65,000 characters, UNICODE currently specifies 38,887
  - Cover languages from Americas, Europe, Middle East, Africa, India, Asia
  - There is space for new characters or application-specific characters
- Problems:
  - Who uses it, yet?
  - More computationally expensive

2. Language identification

- Given a monolingual document from a multilingual collection, determine its language
  - Based on native character encoding (not possible if using UNICODE)
  - Use statistical model of N-grams or words
  - Recognise language-specific characters
  - Use stopwords from IR (Luhn/Zip work)
- More complex if the document is multilingual

3. Segmentation/Tokenisation

- Identify words/token
- Convert document into a “stream of text”
- Easy for English, French, …
- Much more difficult for Asian languages

如果想松懈一下，您可享受
无比的购物乐趣或在连绵
12,500 公里(7,767英里)的洁白
海滩休息，享受清凉的海风。

您也可嬉水取乐，观看那清
澈的海水一阵阵地扑向洁白
的沙滩，或戴上通气管和鸭脚

Segmentation models

- Different approaches:
  - Unique segmentation: decide whether to put the boundary and each point
  - Plausible strings: produce all substrings that might be useful
  - Plausible interpretation: produce all terms that might be implied
- To simplify, let us assume we are dealing with a monolingual (English) collection of documents

4. Phrase identification and named entity recognition

- Enables to identify:
  - Phrases
    - "Database management systems", "Bank of Scotland", "United States of America", "programming language", etc.
  - Named entities
    - "George Bush", "Johnny Walker", "MI5", "September 11th", etc.
- Approaches:
  - Use part-of-speech tagging
  - Use list of named entities
  - Use proximity search (we will see this approach)

5. Term normalisation

- Normalise text to make it easier to compare
  - Lose case
  - Lose punctuation
  - Arrange in alphabetical order
- Problems:
  - Bag of words!
  - Loss of context!

Twinkle, twinkle, little bat.
How I wonder what you're at!
Up above the world you fly.
Like a tea-tray in the sky.

a above at bat fly how i in like little *re* sky *tea* the the *tray* twinkle twinkle up what wonder world you *you*

6. Stop word removal

- Remove words that are poor descriptors
  - Connectives such as ‘and’ ‘but’ ‘because’
  - Articles: ‘the’ ‘an’ ‘a’
  - Prepositions: ‘of’, ‘but’
- And perhaps remove numbers and dates
- List of words removed is known as a stopword list
  - Often created in advance
  - A stopword list is based on word frequency

Zipf distribution

- Words are not evenly distributed
  - Across documents, speech, etc
- If we examine how often words appear
  - A few words appear *very frequently*
  - A medium number of words have *medium frequency*
  - Many words occur *very infrequently*
- They exhibit a **Zipf** distribution

Example of term distribution

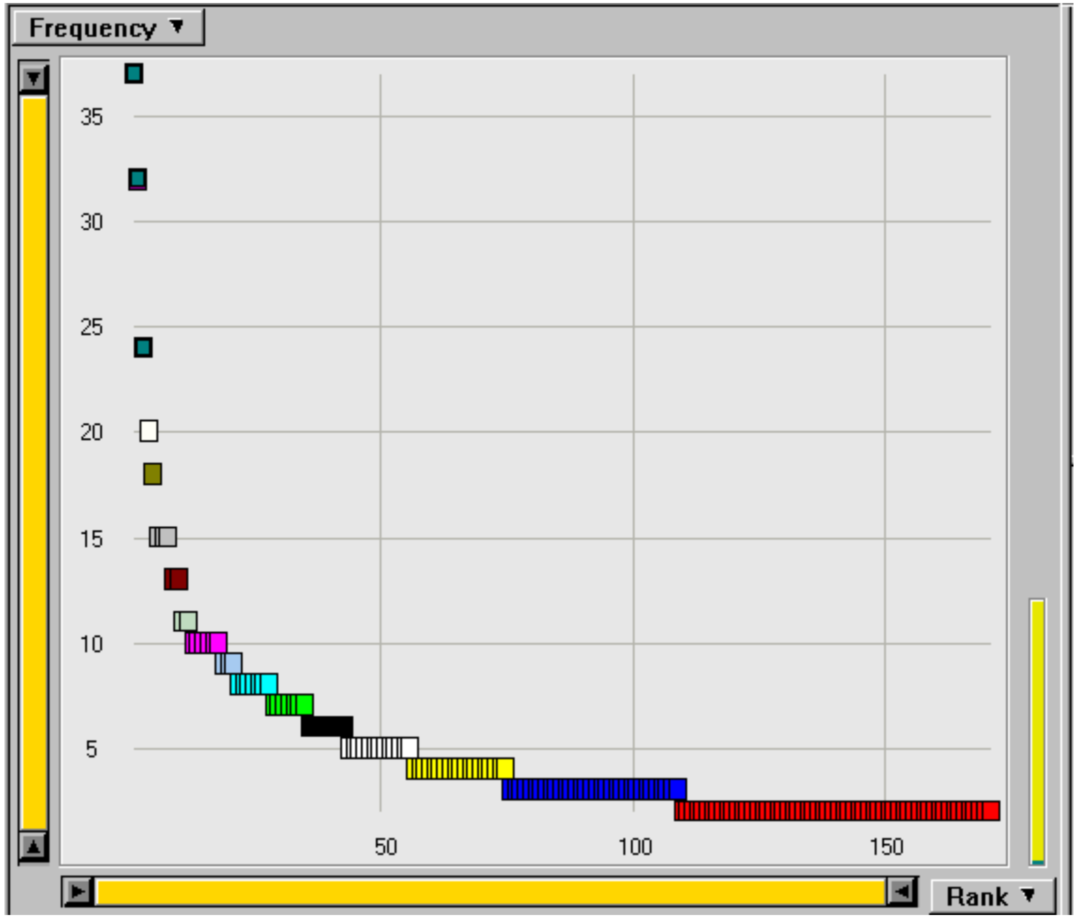
Rank	Freq	Term
1	37	system
2	32	knowledg
3	24	base
4	20	problem
5	18	abstract
6	15	model
7	15	languag
8	15	implem
9	13	reason
10	13	inform
11	11	expert
12	11	analysi
13	10	rule
14	10	program
15	10	oper
16	10	evalu
17	10	comput
18	10	case
19	9	gener
20	9	form

Head and tail of the terms' distribution

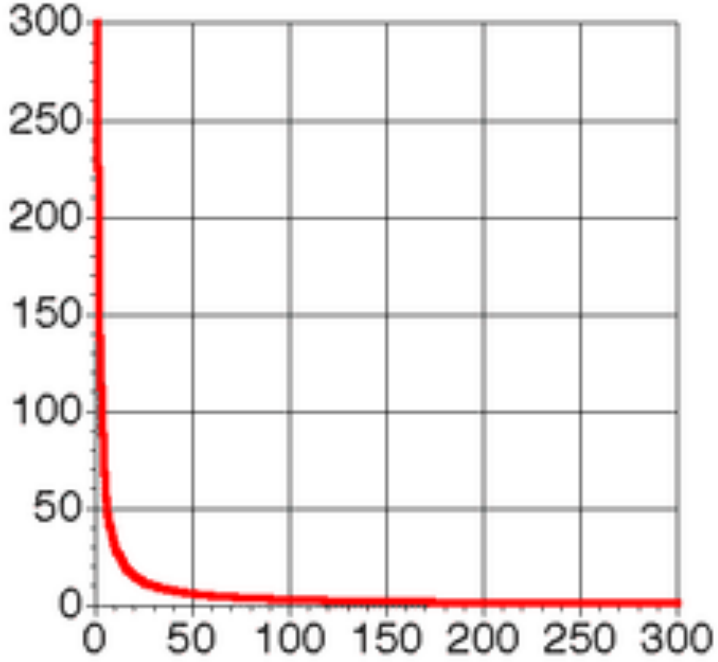
150	2	enhanc
151	2	energi
152	2	emphasi
153	2	detect
154	2	desir
155	2	date
156	2	critic
157	2	content
158	2	consider
159	2	concern
160	2	compon
161	2	compar
162	2	commerci
163	2	clause
164	2	aspect

Corresponding Zipf Curve

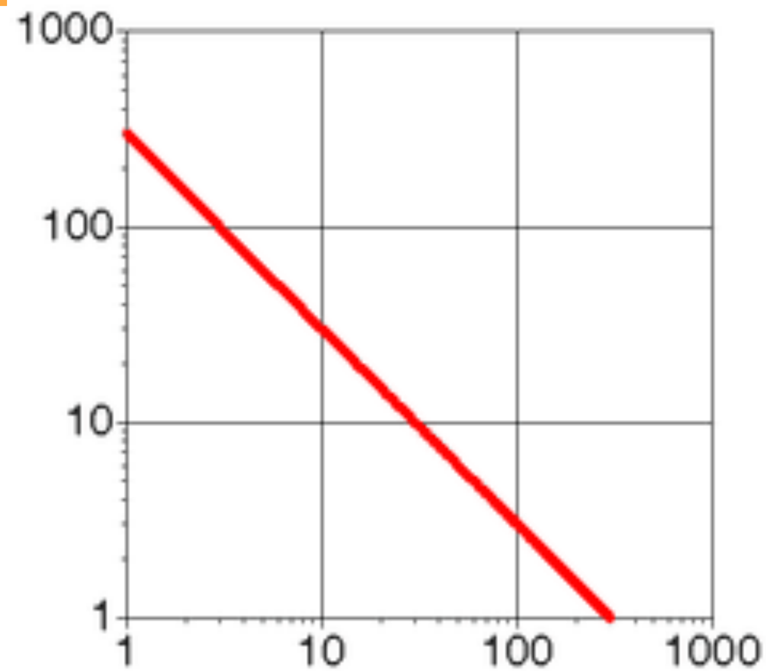
Rank	Freq	Term
1	37	system
2	32	knowledg
3	24	base
4	20	problem
5	18	abstract
6	15	model
7	15	languag
8	15	implem
9	13	reason
10	13	inform
11	11	expert
12	11	analysi
13	10	rule
14	10	program
15	10	oper
16	10	evalu
17	10	comput
18	10	case
19	9	gener
20	9	form



Zipf’s distribution



Raw frequencies



Log of raw frequencies

Zipf distribution

- English (and other languages) follow a Zipf distribution
  - As do other things like website popularity
  - The Zipf distribution is known as “power law”
- High frequency words are useless
  - Describe too many objects and are meaningless
  - These are the stopwords
- Very low frequency words *may* be useless
  - E.g. spelling mistakes, people’s names
  - Too rare to be of value (according to a cost/benefit analysis)
- Best words are middle frequency
  - Used often but not too often

Zipf distribution

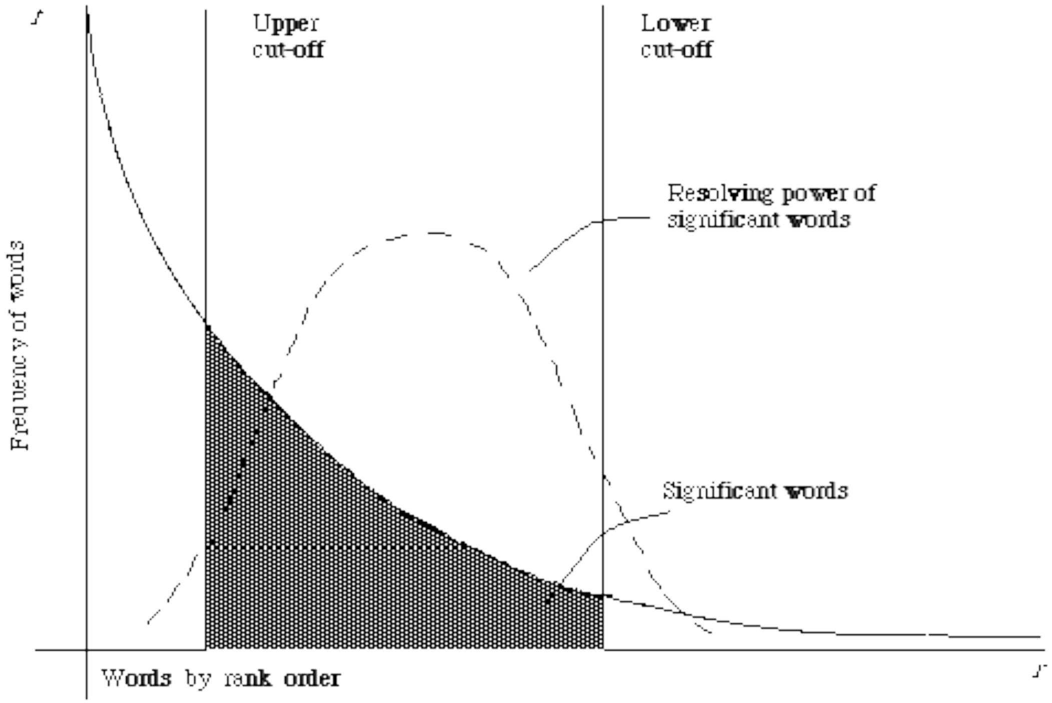


Figure 2.1. A plot of the hyperbolic curve relating f, the frequency of occurrence and r, the rank order (Adaped from Schultz[44] page 120)

Stopword lists

- Common words
  - Standard list contains ~ 300 words
  - a an able about above according accordingly across actually after afterwards again against ….
- Specialised and “ad hoc” stopword lists
  - E.g. remove word that appears in more than 50% of documents
- Stopword removal produces a considerable reduction in the number of words:
  - E.g. WSJ collection (74 520 documents)
    - Without stopword removal - 37 880 008 words
    - With stopword removal – 24 899 830 words
- Results: smaller files and faster search

Our example

twinkle twinkle little bat how i wonder what you re at up high above the world you fly like a tea tray in the sky

25 original words

twinkle twinkle little bat wonder world high like tea tray sky

11 non-stopwords

7. Feature normalisation

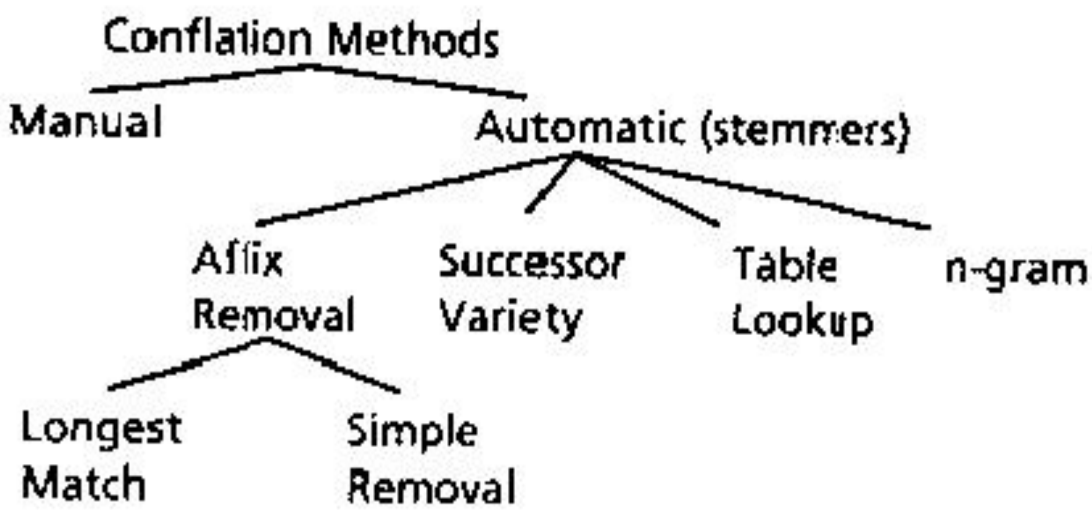
- Words can appear in different forms
- Need some way to recognise common concepts
  - Example:

The Best Hill Walking in Scotland by Cameron McNeish **(1)**	**John Cleare's Fifty Best Hill Walks** by John Cleare **(2)**

- 'Hill walks' will retrieve (2) not (1)
- 'Hill walking' will retrieve (1) not (2)

Stemming

- Stemming is one technique to provide ways of finding morphological variants of search terms
- Used to improve retrieval effectiveness and to reduce the size of indexing files
- Taxonomy of stemming algorithms:



Stemming

- Stem
  - Portion of a word which is left after the removal of its affixes
  - walk ← walked, walker, walking, walks
- Benefits of stemming?
  - Some favor the usage of stemming, but many Web search engines do not adopt any stemming algorithm
- Issues
  - Correctness
  - Retrieval performance
  - Compression performance

Errors generated by stemming

Too aggressive	Too timid
organisation/organ	european/europe
policy/police	cylinder/cylindrical
execute/executive	create/creation
arm/army	search/searcher

Summary

- Original text

Twinkle, twinkle, little bat.
How I wonder what you're at!
Up above the world you fly.
Like a tea-tray in the sky.

- Tokenisation

twinkle twinkle little bat how i wonder
what you re at up high above the world you fly
like a tea tray in the sky

- Stopword removal

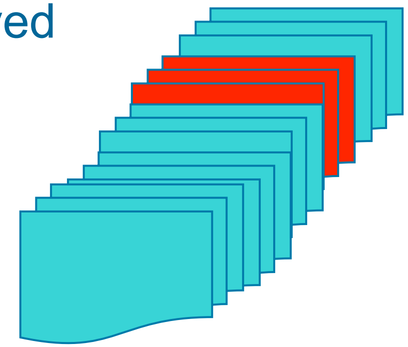
twinkle twinkle little bat wonder
world high like tea tray sky

- Stemming

twinkl twinkl littl bat wonder
world high like tea trai sky

Representing text: considerations

- Two aspects of representation
  - Description
    - What is the content of a document?
    - Important for *recall* - % of relevant material retrieved
  - Discrimination
    - How do I distinguish this document from other documents?
    - Important for *precision* - % of retrieved material relevant
- These act against each other, so a good representation is a balance of both
  - Stopword removal emphasises *discrimination*
    - Reduces the number of words in common between document descriptions
  - Stemming emphasises *description*
    - 'Adds' similar words to documents



8. Term weighting

- Terminology: processed words are known as *index terms*
  - Early IR systems only recorded term *presence* or *absence* (binary weights)
- Example:

docs	twinkl	littl	bat	tea
D1	1	1	1	1
D2	0	0	0	1
D3	0	1	1	1
D4	0	1	0	1
D5	1	0	1	0

Term weighting

- More advanced IR systems weight terms according to *importance*
- Term weighting can be based on frequency of occurrence in:
  - Collection of documents
  - Individual documents
- So, most important term weights:
  - Inverse document frequency (*idf*)
  - Term frequency (*tf*)

Inverse document frequency

- Based on importance of term in the *collection*

$$idf_i = \log\left(N / df_i \right)$$

- N = the number of documents in the collection
- df_i = the number of documents that contain term *t*
- *document frequency* = frequency with which *t* appears

Inverse document frequency

- *idf* gives high values for infrequent terms

- E.g. for a collection of 1000 documents
  - log (1000/1000) = 0
  - log (1000/500) = 0.301
  - log (1000/20) = 2.698
  - log (1000/1) = 4

Example

- idf_{twinkl} =log(5/2) = 0.38
- idf_{littl} =log(5/3) = 0.22
- idf_{bat} =log(5/3) = 0.22
- idf_{tea} =log(5/4) = 0.10

docs	twinkl	littl	bat	tea
D1	0.38	0.22	0.22	0.10
D2	0	0	0	0.10
D3	0	0.22	0.22	0.10
D4	0	0.22	0	0.10
D5	0.38	0	0.22	0

Term frequency

- Based on importance in the *document*
  - Many ways, simplest is raw frequency

$$tf_{di} = num_i$$

  - num_i is the total number of times this term occurs in document *d*

- Sometime it might be useful taking the log of the term frequency

Term frequency

- *tf* gives high values for frequent terms
- E.g. for a document with 11 words
  - "twinkl twinkl littl bat…"

docs	twinkl	littl	bat	tea
D1	2	1	1	1
D2	0	0	0	1
D3	0	1	2	4
D4	0	4	0	5
D5	2	0	3	0

Combined term weighting

- We can combine *tf* and *idf (tf-idf)*

$$weight_{di} = idf_i * tf_{di}$$

docs	twinkl	littl	bat	tea
D1	0.68	0.22	0.22	0.1
D2	0	0	0	0.1
D3	0	0.22	0.44	0.4
D4	0	0.88	0	0.5
D5	0.68	0	0.66	0

TF-IDF normalisation

- Normalize: force all weights to fall within a certain range, usually between 0 and 1, inclusive
- Idea: normalise the term weights
  - Longer documents are not unfairly given more weight)
  - Improves retrieval accuracy

$$weight_{di} = \frac{tf_{di} * idf_i}{\sqrt{\sum_{d=1}^{N} (tf_{di})^2 * idf_i^2}}$$

A collection as a document matrix

- Each document is a vector of term weights and the entire collection can be represented as a matrix

	twinkl	littl	bat	tea	…
D1	0.068	0.02	0.02	0.01	…
D2	0	0	0	0.001	…
D3	0	0.022	0.11	0.002	…
D4	0	0.176	0	0.01	…
D5	0.042	0	0.04	0	…
…	…	…	…	…	…

- $d_i = (t_1, t_2, \ldots, t_n)$

Questions?

A collection as a document matrix

- Each document is a vector of term weights and the entire collection can be represented as a matrix

	twinkl	littl	bat	tea	…
D1	0.068	0.02	0.02	0.01	…
D2	0	0	0	0.001	…
D3	0	0.022	0.11	0.002	…
D4	0	0.176	0	0.01	…
D5	0.042	0	0.04	0	…
…	…	…	…	…	…

- $d_i = (t_1, t_2, \ldots, t_n)$

Storing a document matrix

- We could store documents like this but it would be very wasteful

	twinkl	littl	bat	tea	…
D1	0.068	0.02	0.02	0.01	…
D2	0	0	0	0.001	…
D3	0	0.022	0.11	0.002	…
D4	0	0.176	0	0.01	…
D5	0.042	0	0.04	0	…
…	…	…	…	…	…

Document matrix

- Document matrices are very sparse
  - E.g. WSJ (1990-93)
    - 74 520 documents 123 852 unique words
    - Average doc length: ~300 words
    - Average occurrence: ~200 docs
    - After indexing only 0.00065% of cells of the document matrix are filled
- We need a different way to store the information contained in the matrix

Index data structures

- Index data structures enable space efficient storage of document content descriptors
- The most common index data structure in IR is the *inverted index*
- From an inverted index we can build *inverted files*
- Inverted files are the most common data structure for efficient storage and fast processing of IR indexes

Inverted index

- An "inverted index" is a vector index "inverted" so that rows become columns and columns become rows

docs	*t1*	*t2*	*t3*
D1	1	0	1
D2	1	0	0
D3	0	1	1
D4	1	0	0
D5	1	1	1
D6	1	1	0
D7	0	1	0
D8	0	1	0
D9	0	0	1
D10	0	1	1

Terms	D1	D2	D3	D4	D5	D6	D7	…
t1	1	1	0	1	1	1	0	
t2	0	0	1	0	1	1	1	
t3	1	0	1	0	1	0	0	

Inverted index files

- Documents are tokenised, stopwords
- removed, and stemmed
- Tokens saved with document identifier

“Be vewy vewy quiet. Today, I am hunting wabbits” (Doc 1)

“I thought I saw a puddy cat. I did, I did I saw a puddy cat” (Doc 2)

Term	DocID	#terms
be	1	9
vewy	1	9
vewy	1	9
quiet	1	9
today	1	9
i	1	9
am	1	9
hunting	1	9
wabbits	1	9
i	2	16
thought	2	16
I	2	16
saw	2	16
a	2	16
puddy	2	16
cat	2	16
…	…	…
cat	2	16

Index files

- Table is sorted alphabetically

Term	DocID	#terms
be	1	9
vewy	1	9
vewy	1	9
quiet	1	9
today	1	9
i	1	9
am	1	9
hunting	1	9
wabbits	1	9
i	2	16
thought	2	16
I	2	16
saw	2	16
a	2	16
puddy	2	16
cat	2	16
…	…	…
cat	2	16

Term	DocID	#terms
a	2	16
a	2	16
am	1	9
be	1	9
cat	2	16
cat	2	16
did	2	16
did	2	16
hunting	1	9
i	1	9
i	2	16
i	2	16
i	2	16
i	2	16
i	2	16
puddy	2	16
…	…	

Index files

- Multiple entries for each document are merged
- Within-document frequencies (*tf*) values are calculated

Term	DocID	*tf*
a	2	0.125
am	1	0.111
be	1	0.111
cat	2	0.125
did	2	0.125
hunting	1	0.111
i	1	0.111
i	2	0.313
puddy	2	0.125
…	…	

Index files

- From this we create two files:
  - *Dictionary file*: list all terms in the collection with their global term weights
  - *Postings file*: list the occurrences of all terms in each document, indicating also the local term weight

Dictionary file

- Collection information
  - *idf* weights

No of docs in which term appears

***idf* weight**

Where to start reading postings file

Term	# docs	*idf*	offset
a	1	0.301	0
am	1	0.301	1
be	1	0.301	2
cat	1	0.301	3
did	1	0.301	4
hunting	1	0.301	5
i	2	0.000	6
puddy	1	0.301	8
…	…	…	

Offset

- Offset can count
  - Tuples to read
  - Numbers to read
  - Bytes to read

2 0.125 1 0.111 1 0.111 2 0.125 2 0.125 1 0.111 1 0.111 2 0.313 2 0.125 …

Postings file

- Which documents contain a term
  - Reduces sparse nature of data
- Series of tuples <docID, *tf* weight>

2 0.125 1 0.111 1 0.111 2 0.125 2 0.125 1 0.111 1 0.111 2 0.313 2 0.125 …

term 'a' appears in document 2 with *tf* 0.125

term 'i' appears in document 1 with *tf* 0.111 *and* document 2 with *tf* 0.313

Example

- Query ‘hunting cat’

Term	# docs	*idf*	offset
a	1	0.301	0
am	1	0.301	1
be	1	0.301	2
cat	1	0.301	3
did	1	0.301	4
hunting	1	0.301	5
i	2	0.000	6
puddy	1	0.301	8
…	…	…	

2 0.125 1 0.111 1 0.111 2 0.125 2 0.125 1 0.111 1 0.111 2 0.313 2 0.125 …

Index files summary

- Index files mean fast access
  - Postings files reduce data size
    - And only store which terms appear in a document
    - And document specific information
  - Dictionary gives collection information
    - And tells us how to read postings file
- Index files can be distributed across several machines, be partitioned and can be accessed in parallel

Advanced indexing: word position

- Store word positions
  - Standard postings file

“I thought I saw a puddy cat. I did, I did I saw a puddy cat”
(Doc 2)

2 0.125 1 0.111 1 0.111 2 0.125 2 0.125 1 0.111 1 0.111 2 0.313 2 0.125 …

  - With word positions

Terms ‘a’ appears in document 2 with *tf* 0.125 and appears at positions 5 and 14

2 0.125 5 14 1 0.111 7 …

  - Closer query words in document appear better match

Advanced indexing: word position

- Storing the word positions enables to search and retrieve documents using exact phrases or named entities:
  - "Bank of Scotland" and not "on the bank of the Lock Lomond in Scotland"
  - "George Bush" and not "George fall on the bush"
  - "University of Strathclyde" and not "University of Glasgow, Glasgow, Strathclyde"

Advanced Indexing: document expansion

- Add semantics by "expanding" the document representations by using knowledge structures:
  - Dictionaries
  - Thesauri
  - Ontologies
- This enables to retrieve a document even if a word does not appear, but some synonym or closely related word appears

Advanced Indexing: document expansion

- E.g. if document contains word 'book' add related meanings
  - album, atlas, bestseller, bible, **book**let, brochure, codex, compendium, copy, dictionary, dissertation, edition, encyclopedia, essay, fiction, folio, hand**book**, hardcover, leaflet, lexicon, magazine, manual, monograph, nonfiction, novel, octavo, offprint, omnibus, opus, opuscule, pamphlet, paperback, periodical, portfolio, preprint, primer, publication, quarto, reader, reprint, scroll, softcover, speller, text, text**book**, thesaurus, tome, tract, treatise, volume, work, writing
- But there might be problems of ambiguity
  - Book = reserve (v), volume (v)
- And reduces discrimination

Query indexing

- Queries are indexed too
    - *"I want information on the semiotic importance of Daffy Duck and Daffy's role in the political hagiography of Elmer Fudd"*
  - Stopword removal
    - *"information semiotic importance daffy duck daffy role political hagiography elmer fudd"*
  - Stemming
    - *"inform semiot import daffy duck daffy role polit hagiograph elmer fudd"*
  - Term weighting
    - *informat* 0.09, *semiot* 0.09, *import* 0.09, *daffy* 0.18,…
    - Not usually done

- Query and documents use same representation, so it is easier to carry out matching

Retrieval

- How do we find relevant documents?

docs	twinkl	littl	bat	tea
D1	0.68	0.22	0.22	0.1
D2	0	0	0	0.1
D3	0	0.22	0.44	0.4
D4	0	0.88	0	0.5
D5	0.68	0	0.66	0

Q1 = “twinkle littl”
Q2 = “tea bat”

	Q1	Q2
D1	?	?
D2		
D3		
D4		
D5		

- NEXT!

Questions?

Retrieval: outline

- How do we formulate IR queries?
- How do we evaluate the relevance of a document to a query?
  - Retrieval models
- Relevance feedback
  - Methods for automatically modifying user query
  - Probabilistic model (the probability estimation requires users feedback)
- Results presentation
  - Elements of interfaces for IR systems

Last time

- How do we find relevant documents?

Q1 = “twinkle littl”
Q2 = “tea bat”

docs	twinkl	littl	bat	tea
D1	0.68	0.22	0.22	0.1
D2	0	0	0	0.1
D3	0	0.22	0.44	0.4
D4	0	0.88	0	0.5
D5	0.68	0	0.66	0

	Q1	Q2
D1	?	?
D2		
D3		
D4		
D5		

Retrieval models

- A retrieval model is a mathematical model that enables to associate a value (score) to a pair (d, q)
  - Often called retrieval status value (RSV)
- This value is an estimate of the relevance of d to q
- Depending on the model the RSV has different interpretations
- Three major retrieval models
  - Boolean model
  - Vector-space model
  - Probabilistic model

Boolean retrieval

- Oldest model
- Based on Boolean logic
- Terms and connectors
- Terms – query words ‘*cat*’ ‘*house*’ ‘*bat*’
- Connectors
  - AND, OR, NOT
  - Similar to structured language

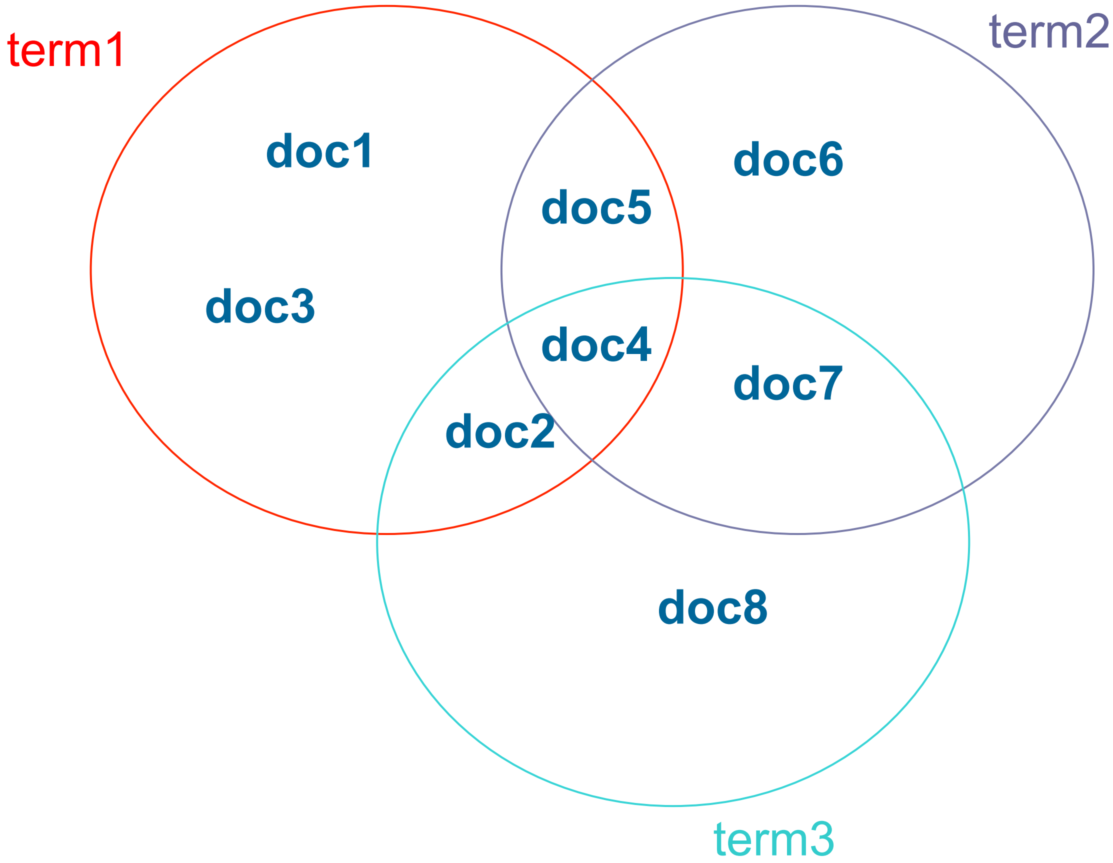
Example Boolean queries

- cat
  - documents containing 'cat'
- cat OR dog
  - docs containing 'cat' or docs containing 'dog'
- cat AND dog
  - docs containing 'cat' and 'dog'
- (cat AND dog) OR budgie
  - docs containing 'cat' and 'dog' or docs containing budgie
- NOT budgie
  - docs that do not contain budgie
- NOT ((cat AND dog AND budgie) OR (cat AND budgie) OR (cat AND dog) OR (cat))
  - ????

Boolean queries

- Usually expressed as infix operators
  - ((a AND b) OR (c and b))
- NOT is a prefix operator
  - NOT (b), (c AND (NOT (b)))
- AND and OR are n-ary
  - (a AND b AND c)
- Heavy use of rules
  - NOT (a) AND NOT (b) = NOT (a OR b)
  - NOT (a) OR NOT (b) = NOT (a AND b)
  - NOT (NOT (a)) = a

Boolean logic



Boolean retrieval

- Typically no term weighting
- Run a query, get a result set
  - Unordered
  - Popular terms – big result set
  - Rare terms – small result set
  - AND leads to big result set
  - OR leads to small result set
  - Bad combination – huge result set or empty result set

Wrong set size

- Two choices
  - Run new query on entire collection
  - Run modified query on results set
- Example:
  - (redford AND newman) -> S1 1450 documents
  - S1 AND Sundance -> S2 898 documents

Advantages/disadvantages

- Advantages
  - Complete expressiveness
  - Exact queries
  - Simple to program
  - Boolean algebra
    - experts
- Disadvantages
  - Artificial language
    - Unintuitive
    - Misunderstood
  - Too many, too few results
  - Unordered output
    - Date at best

Extensions

- Trying to overcome poor points
  - Use of term weights
  - Proximity search
  - Filters
  - User interfaces

Proximity search

- Proximity: terms occur within K positions of each other
  - pen w/5 paper
    - (a) "pen and paper" matches
    - (b) "my pen is on my desk next to my paper" does not match
  - So need to store position in postings file
  - A "near" function can be more vague
    - near (pen, paper) – both (a) and (b) match
  - Phrases – "Bill Clinton"
    - Phrase variant – "information retrieval" "retrieval of information"

Filters

- Reduce set of candidate documents
  - Restrictions on documents
    - Date range
    - Internet domain (.uk, .com, .strath.ac.uk)
    - Author
    - Size
    - Limit number of documents returned

Partial-match models

- Ranked output
  - Documents ranked according to how closely they match query
- Advantages over Boolean
  - No query syntax
    - Natural language
    - Easier to modify query
  - Documents can partially match the query
    - Easier to interpret results

Simplest way to rank documents

$$sim(q, d_j) = \sum_{i=1}^{n=|q|} w_{ij}$$

- Query – *littl bat*

docs	twinkl	littl	bat	tea	*sim*
D1	0.054	0.09	0.36	0.242	0.45
D2	0	0	0	0.027	0.00
D3	0	0.1	2	0.005	2.10
D4	0.031	0.08	0	0.297	0.08
D5	0	0	0.08	0	0.08

Vector-space model

- 1960's mostly by G. Salton
- Very influential model
- Documents and queries are *vectors*
- Simple example:

docs	Term1	Term2
D1	0.5	1
D2	1	0
D3	0	0.5
D4	0.25	0.25

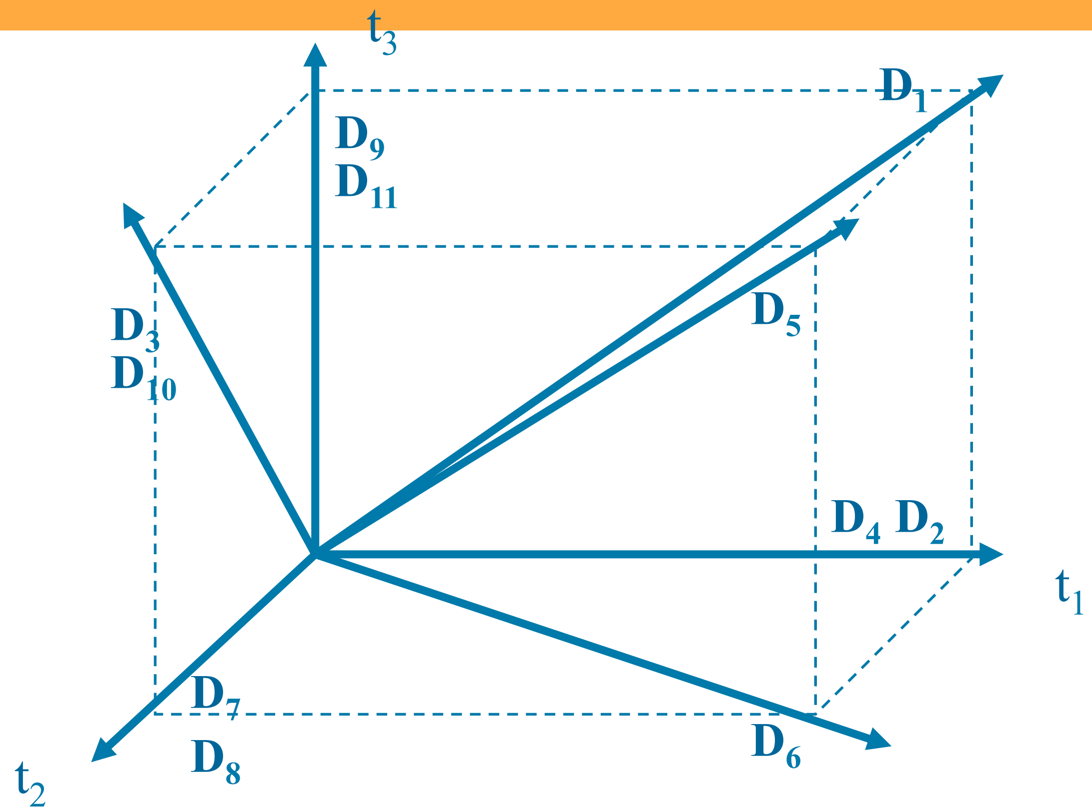
D1 = <0.5,1>

D2 = <1,0>

D3 = <0,0.5>

D4 = <0.25,0.25>

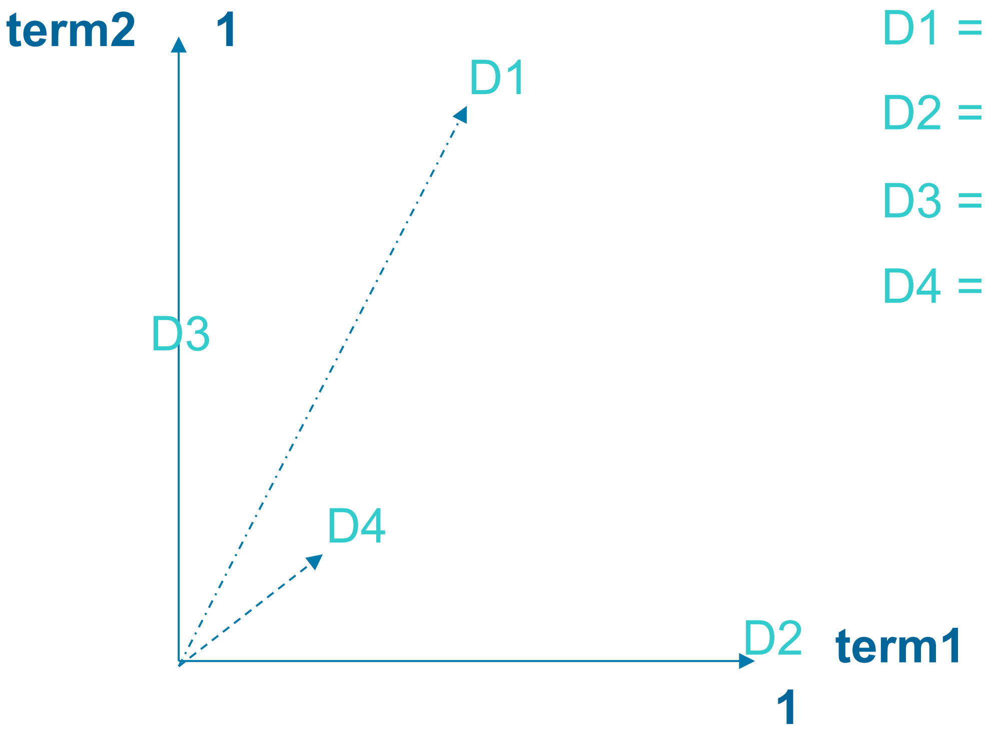
A collection as a vector space



More about this later!

Vectors

- Vectors define a position in space
  - Size of vectors = number of words in collection



D1 = <0.5,1>

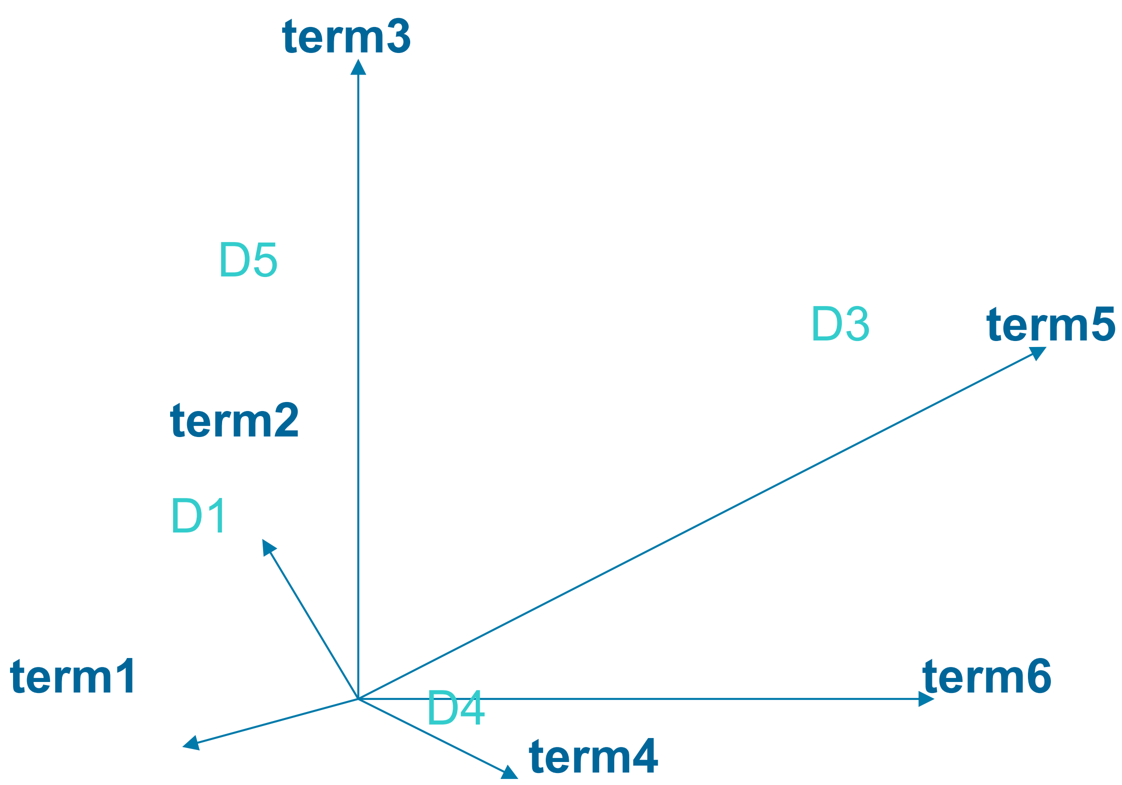
D2 = <1,0>

D3 = <0,0.5>

D4 = <0.25,0.25>

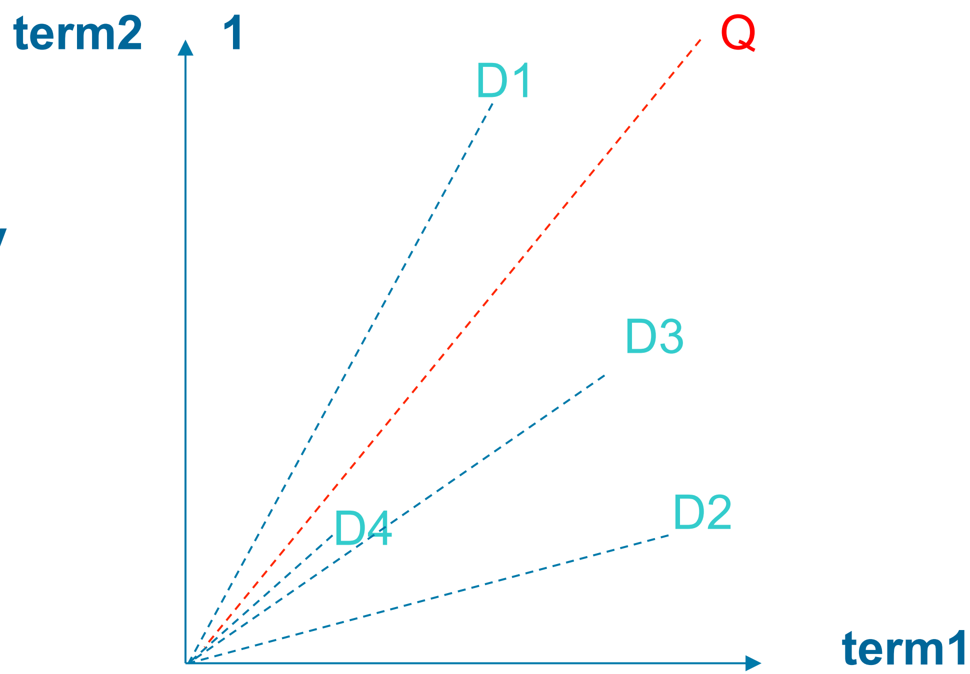
More complicated example

- Each term is an axis
  - Position on axis = weight of term in document



Retrieval

- The ‘closer’ two documents are in space, the more similar they are
- The closer a query is to a document, the better the query matches



Similarity measures

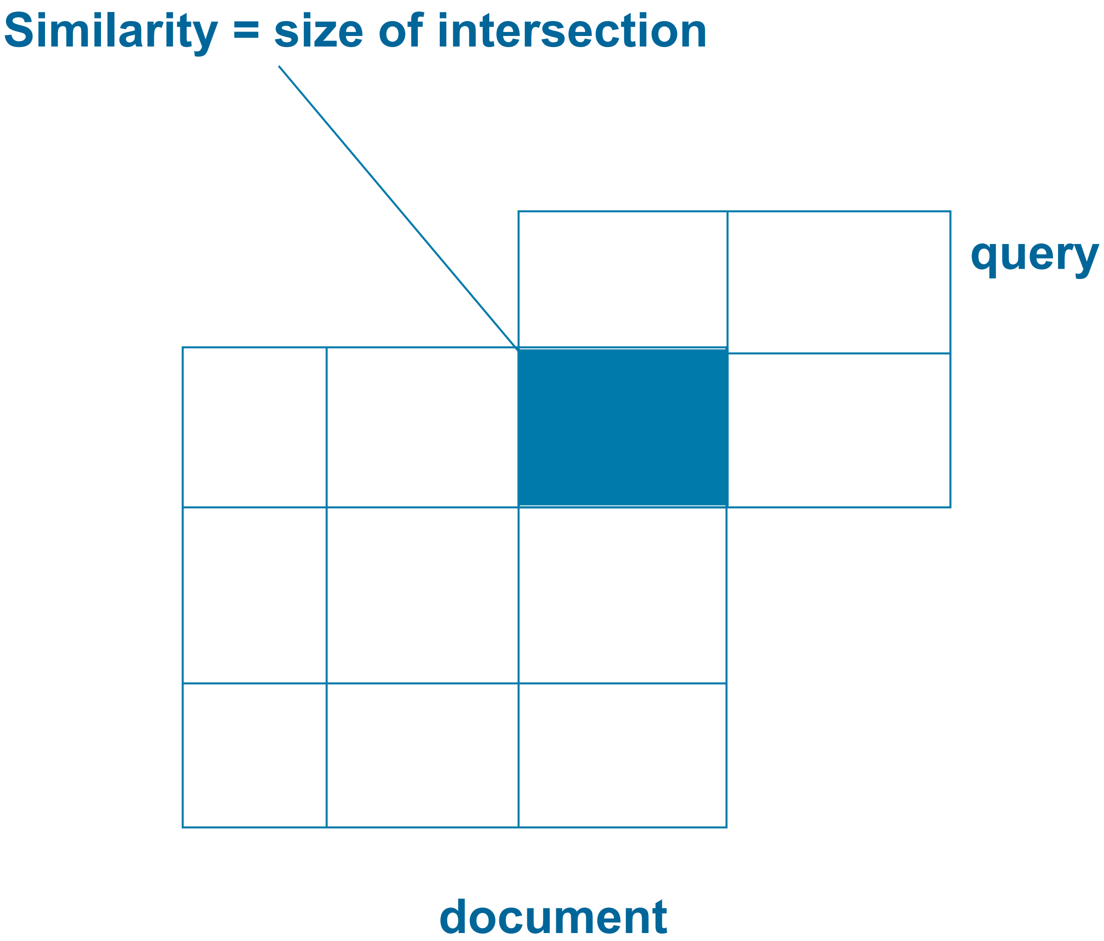
- Simple matching (coordination level)

$$RSV(q,d) = \left|d \cap q\right|$$

- Cosine correlation coefficient
  - (most important)

$$RSV(q,d) = \frac{\left|q \cap d\right|}{|q|^{0.5} * |d|^{0.5}}$$

Simple matching

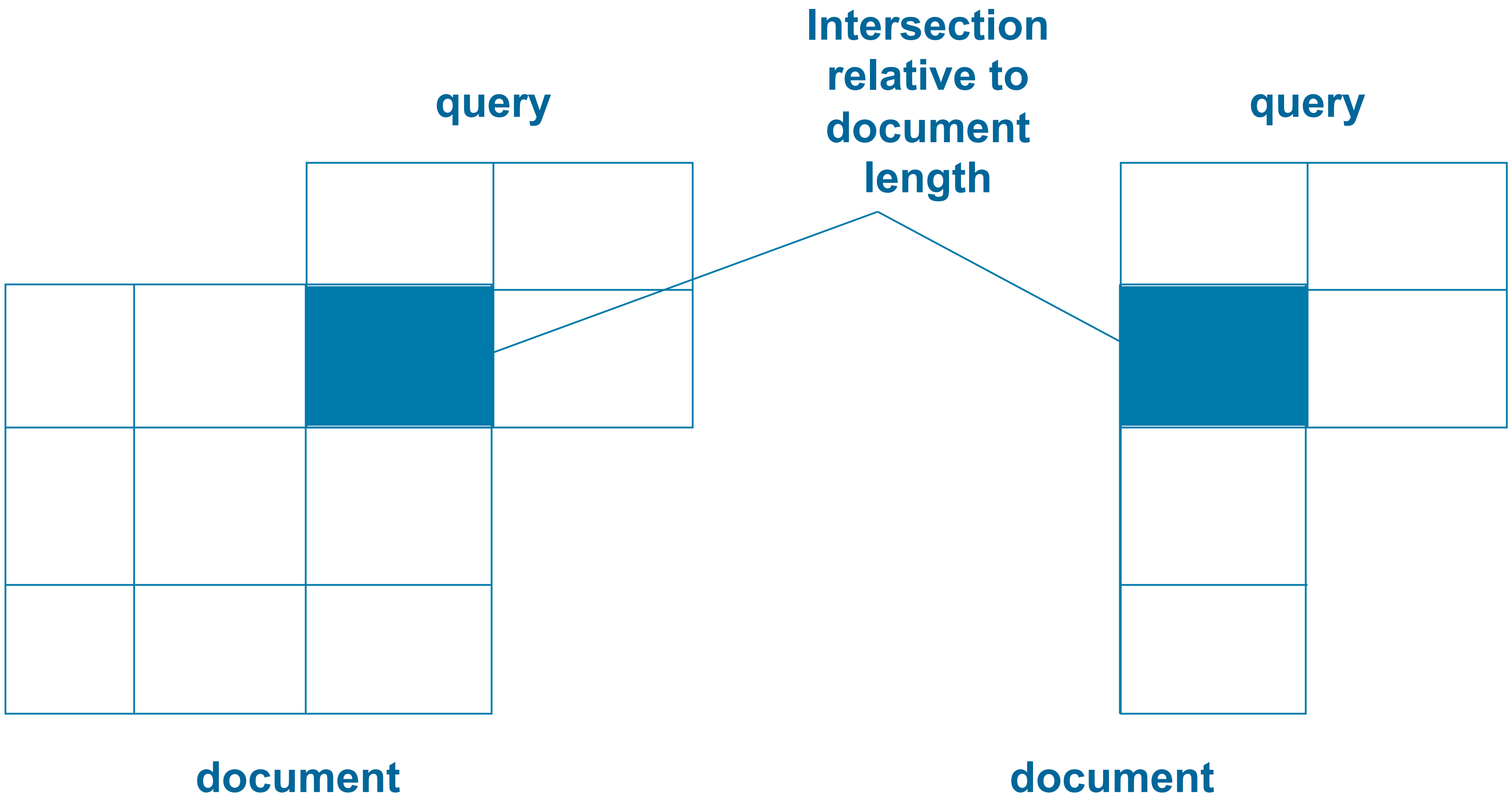


Simple matching

- Simple matching
  - Intersection of terms in query and document
  - Higher intersection
    - More terms in common
  - If terms have weights
    - Share more higher weighted terms

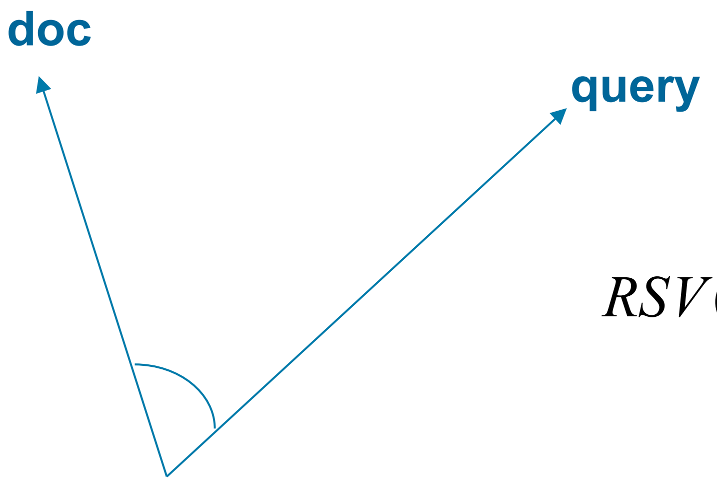
$$RSV(d,q) = \sum_{i=1}^{n} \left(w_{di} * w_{qi} \right)$$

Problems with simple matching



Cosine matching

- Cosine correlation matching
  - Matching cosine of angles
  - Bigger difference, smaller value
    - E.g. cosine(0) = 1, cosine(180) = -1
  - Take into consideration the length of the vectors



$$RSV(d,q) = \frac{\sum_{i=1}^{n}\left(w_{di} * w_{qi}\right)}{\sqrt{\sum_{i=1}^{n}\left(w_{d_i}\right)^2} * \sqrt{\sum_{i=1}^{n}\left(w_{q_i}\right)^2}}$$

Advantages/disadvantages

- Advantages
  - Easy to understand
  - Has a geometric interpretation
  - Works generally well with any similarity measure
  - Better performance than Boolean (mostly)
- Disadvantages
  - No real theoretical basis
  - Easy to modify with ad-hoc tricks
  - Dimensions are not orthogonal
    - terms are not independent
  - Less control?

Questions?

Probabilistic model

- It is a best-match retrieval model
  - Rank documents for presentation to user
    - Vector-space uses geometric analogy
    - Probabilistic model tries to evaluate probability of observing relevance given a specific pair (q, d)
  - Query in natural languages
    - No query syntax
- The probabilistic model has implicit the notion of relevance feedback
  - Relevance feedback is a query reformulation technique

Relevance feedback

- Motivations
  - Queries can be difficult to create
    - User's often don't know what they want
  - Verbalising a query can be hard
    - But recognising relevant information is usually easier
- Relevance feedback
  - Showing system what you want
  - System modifying your query

Relevance feedback

- What it’s trying to do
  - Use examples of documents the user likes to
    1. Detect which words are useful
       - New query words
       - Query expansion
    2. Detect how useful these words are
       - Change weights of query words
       - Term re-weighting
    3. Use new query for retrieval

Query expansion

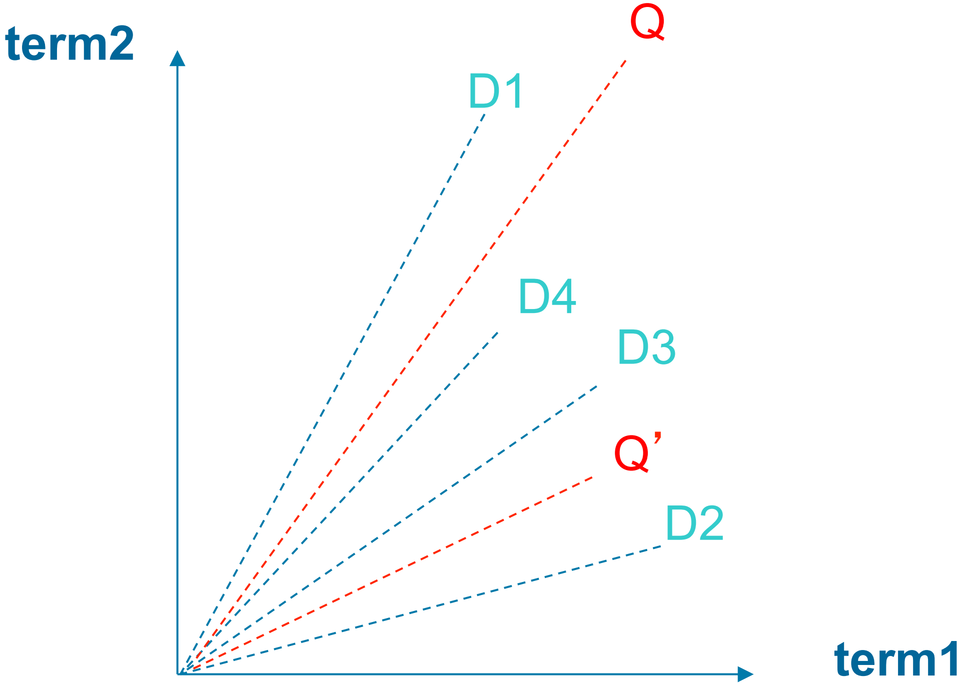
- Add useful terms to the query
  - Terms that appear often in relevant documents
  - Trying to compensate for poor queries
    - Usually short, can be ambiguous, use too many common words
    - Add better terms to user’s query
  - Trying to emphasise *recall*

Term re-weighting

- Re-weight query terms
  - Start off with weights derived from query
    - E.g. "daffy duck" daffy 1 duck 1
    - E.g. "daffy daffy duck" daffy 2 duck 1
  - Assign new weights according to importance in relevant documents
  - Personalised searching
    - Which query terms are important to the user
  - Trying to improve *precision*

Vector space model

- D2, D3 relevant
- D1, D4 not relevant
  - Aim: make Q vector closer to D2,D3 and further from D1,D4
  - Result: Q’ is the new query vector



Advantages of relevance feedback

- Advantages:
  - RF means altering the user’s query
  - Can be very effective
  - Breaks down search into chunks, gradually improving the query
  - Less emphasis on query, more on documents
- Disadvantages:
  - Relevance is binary for systems, but not for users
  - Only parts of documents may be relevant
  - No feedback to users
    - How does it work?

Probabilistic model

- Alternative to vector-space
  - W. Maron, S. Robertson , K.Sparck Jones, C. J. van Rijsbergen and others (1960s onwards)
  - Designed specifically for relevance feedback
- Estimate probability of relevance
  - Observing relevance given a pair (q, d)
  - Use terms as evidence
  - Estimate probability that a query term will appear in a relevant document
  - Re-weights query terms using relevance information

Probabilistic model

- Assign new weights for query terms based on relevant/non-relevant documents
- Give higher weights to important terms:

	Relevant	Not-relevant	
Documents contain term	*r*	*n-r*	*n*
Documents do not contain term	*R-r*	N-n-R+r	*N-n*
	R	*N-R*	

Probabilistic term re-weighting formula

Relevant documents with *t*

Relevant documents without *t*

$$w(t) = \log\left(\frac{\dfrac{(r_t + 0.5)}{(0.5 + R - r_t)}}{\dfrac{(0.5 + n_t - r_t)}{(0.5 + N - n_t - R + r_t)}}\right)$$

Non-relevant documents with *t*

Non-relevant documents without *t*

Document score based on sum of weights of query terms in documents

Probabilistic model

- Advantages over vector-space
  - Strong theoretical basis
    - based on probability theory (very well understood)
    - easy to extend
- Disadvantages
  - Models are often more complicated than vector space models
  - No term frequency weighting
- Which is better vector space or probabilistic?
  - Both are approximately as good as each other

Relevance feedback

- Problems with RF and users
  - Relevance is binary for systems
    - But not for users
  - Only parts of documents may be relevant
    - Sentence, paragraph, title, …
  - No feedback to users
    - How does it work?
    - What does it mean?

Questions?

Topics still missing

- More advanced models (e.g. language models, topics models, logical models, …)
- Evaluation of IR Systems
- Indexing and retrieval of social media and multimedia
- Many more …

Conclusions

- Very fast introduction to IR!
- Hope it gave you an understanding of the many issues involved
- Should enable you to understand the remaining lectures
- For more information see one of the many textbooks in IR that are currently available!

Thank you for listening!

- Last chance to ask questions …