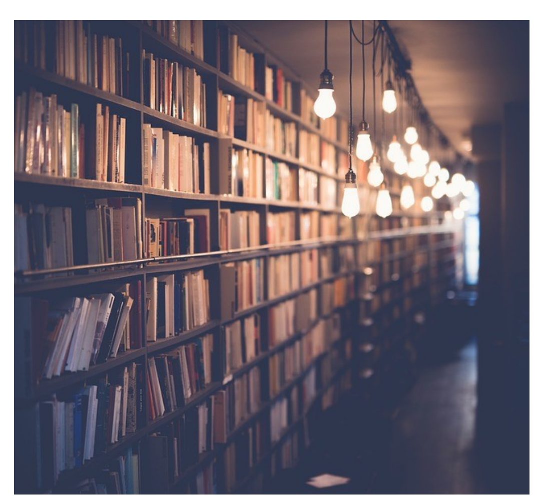
Information retrieval

Structured vs. unstructured content

- Structured content
  - Relational databases (SQL)
  - NoSQL databases (JSON)
- Searching unstructured content in large databases
  - Text
  - Images
  - Video
  - Sound

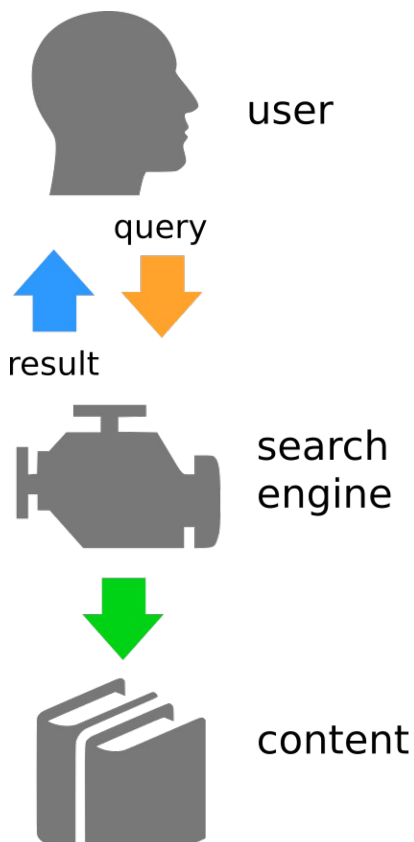


Overview

- Search using Boolean expressions
- Building search dictionary
- Approximate queries
- Ranking results
- Document similarity
- Feedback loops
- Retrieval sysems evaluation

Conceptual model

- Content retrieval steps
  - User submits a query
    (how to use language to specify what we are looking for?)
  - Compose and rank results based on the data
    (how to match query with documents?)
  - User evaluates results
    (how to optimize the query for better experience?)
- Iteration of steps can improve results quality



Example of search engine

user

query

free text
query parser

spelling correction

documents

parsing & linguistics

indexers

Tiered positional index

k-gram

Metadata

Inexact top K retrieval

scoring and ranking

result

machine-learned relevance

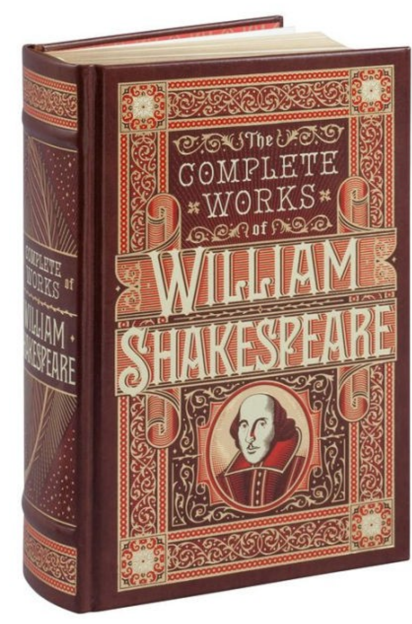
training data

Search engine challenges

- Data is unstructured, not suitable for direct retrieval
- Multiple ways to set up the same query – describe the requested information
- Large quantities (data and queries)
- Two key processes: Querying and Indexing

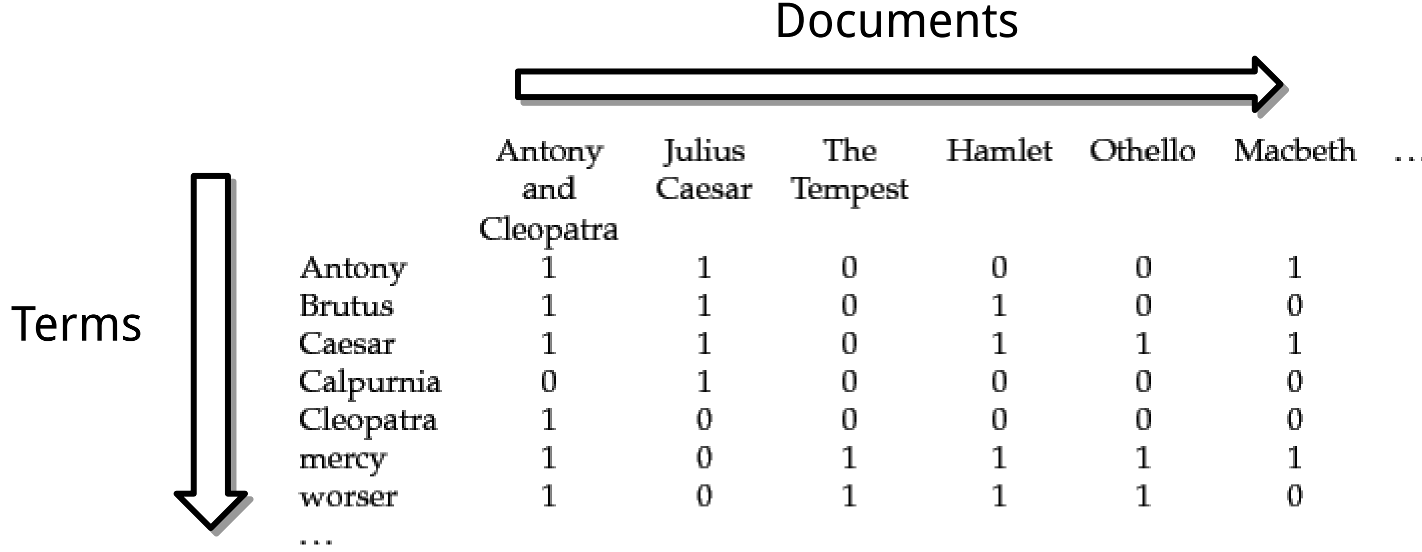
Searching using Boolean expressions

- Shakespeare – The Complete Works
- Which plays contain words Brutus and Caesar, but do not contain word Calpurnia?
- Naive approach:
  - Sequentially scan text of all plays
  - Takes a lot of time (especially on large databases)
- Better approach: pre-index all documents



Incidence matrix

For each term remember which documents contain it



	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Antony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
...							

Retrieval example

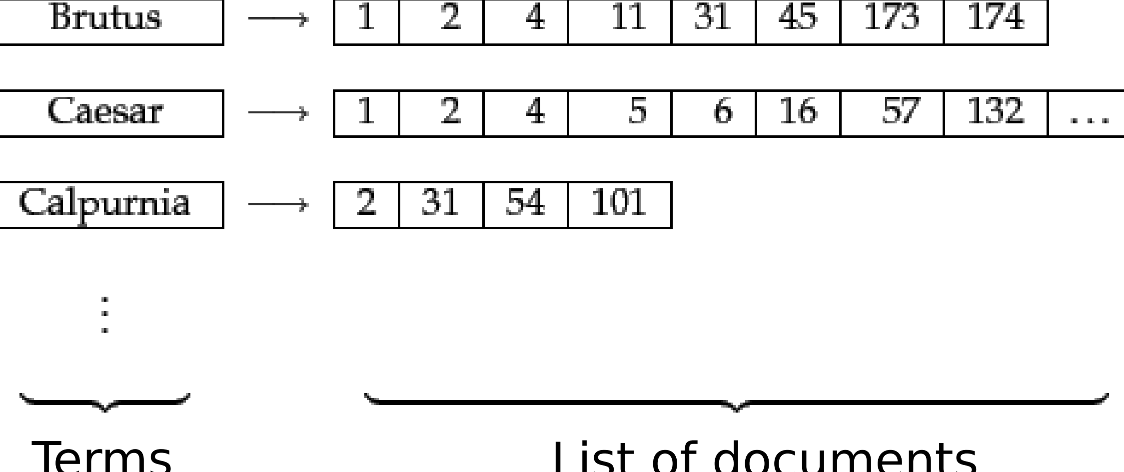
- Query: »Brutus AND Caesar AND NOT Calpurnia«
- Obtain binary vectors for all three terms, negate the last one and join them with AND:

		Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
-	Antony	1	1	0	0	0	1	
1	Brutus	1	1	0	1	0	0	
1	Caesar	1	1	0	1	1	1	
0	Calpurnia	0	1	0	0	0	0	
-	Cleopatra	1	0	0	0	0	0	
-	mercy	1	0	1	1	1	1	
-	worser	1	0	1	1	1	0	
...	...							
		1	0	0	1	0	0	

- Limitation: computer memory

Inverted index

- Incidence matrix is sparse
- Per-term list of documents that include the term



Processing queries

- Query: »Brutus AND Calpurnia«
- Find lists of documents that include »Brutus« and »Calpurnia«
- Compute intersection of lists (linear complexity for ordered lists)

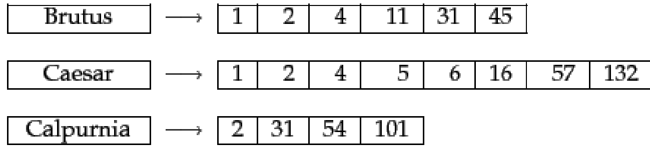
Brutus ⟶ 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

Calpurnia ⟶ 2 → 31 → 54 → 101

Intersection ⟹ 2 → 31

Optimizing queries

- **AND query**: »Brutus AND Caesar AND Calpurnia«
  - Reduce number of comparisons – start with two least frequent terms



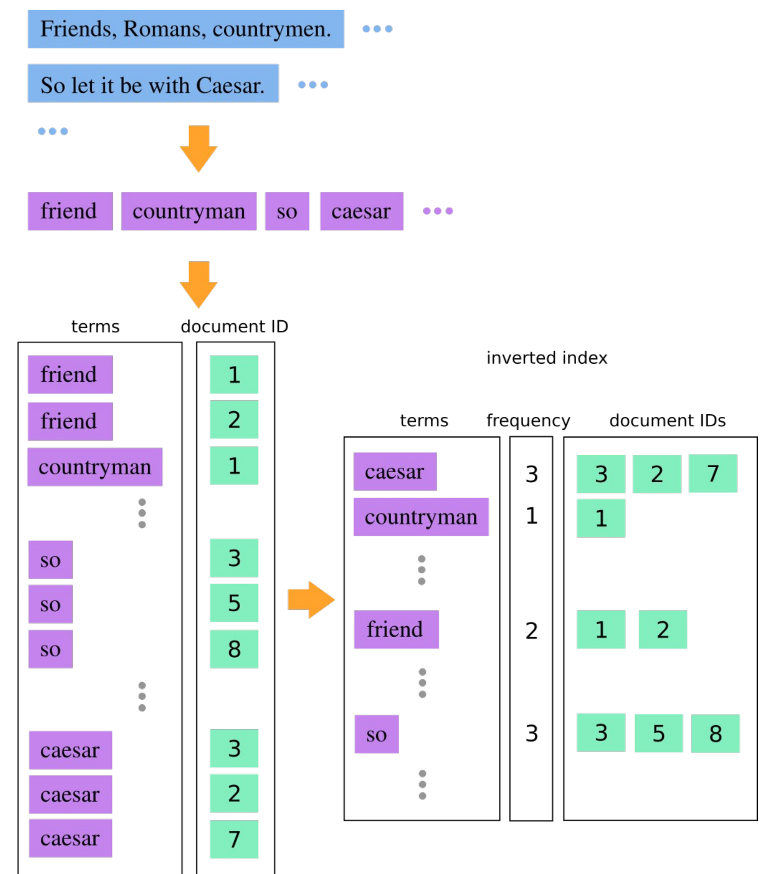
  - Optimized: »(Calpurnia AND Brutus) AND Ceasar«
- **OR query**: »(madding OR crowd) AND (killed OR slain)«
  - Sum terms in OR relation to obtain conservative estimates of combined lists
  - Sort AND queries based on the estimates

Choosing the »document« unit

- Granularity
  - Are documents files? (MS Word, LibreOffice, ...)
  - What about mailbox file full of emails? (Thunderbird, Outlook)
  - Attachments in email messages?
- Fine-grained – bad recall of relevant documents
- Coarse-grained – recall opacity
- Document selection depends on the use-case
- We can also search at multiple granularity levels

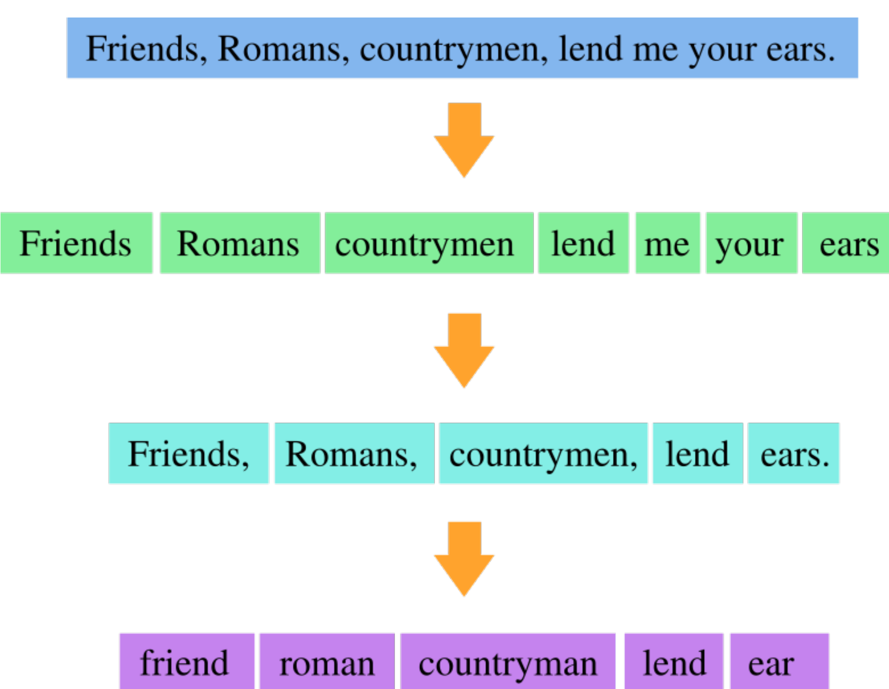
Building inverted index

- Split each document into a list of tokens
- Linguistic processing, tokens normalization
- Build a list of (token, document) pairs
- Sort the list alphabetically by token
- Group occurrences of same token into list
- Remember document frequency



Building term dictionary

- Tokenization (sequence to tokens)
- Exclude »stop« tokens/words
- Normalization (equivalence classes)
- Stemming and lemmatization



Decoding content

- Text is a sequence of bytes
- Different encoding schemes: ASCII, UTF-8, ...
- Is text a linear, unambiguous sequence of characters?

كِتابٌ ⇐ ٌ ب ا ت ِ ك

un b ā t i k

/kitābun/ *'a book'*

استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.

← → ← → ← START

'Algeria achieved its independence in 1962 after 132 years of French occupation.'

- Other modalities even more complicated (e.g. images)

Tokenization

- Punctuation marks:
  - U.S.A = USA , O'Niel = Oniel
  - Problems: C.A.T. = CAT ?? ... Civil Air Transport (C.A.T)
- Connected words:
  - lower-case = lowercase,
  - San Francisco = SanFrancisco
  - San Francisco-Los Angeles = ?
  - Lebensversicherungsgesellschaftsangestellter = ?
- Numbers:
  - (800) 234-2333, (Mar 11 1983), (3/11/1983)
- More language-specific definitions (East Asia – no spaces)

Removing stop words

- Words that occur very often in all documents and therefore do not have any retrieval value

a	an	and	are	as	at	be	by	for	from
has	he	in	is	it	its	of	on	that	the
to	was	were	will	with					

- How to query “Let It Be“ or “The Who”?
- Some search engines do not use stop words to support phrase search

Token normalization

- Removing accent marks (diacritics)
  - cliché = cliche
  - peña = pena
  - Universität = Universitaet
- Convert to lower-case
  - Father = father
  - General Motors = general motors (company name - phrase)
- Language specific conversions
  - colour = color
  - 30.10.1978 = 10.30.1978

Stemming and lemmatization

- Lemmatization – transformation based on language rules
  - am, are = be
  - car, cars, car's, cars' = car
  - »the boy's cars are different colors« = »the boy car be differ color«
- Stemming is heuristic approach where we only cut parts of words (faster)
  - Porter Stemming Algorithm
  - »boy ’ s car are differ color«

Querying phrases

- How to search for »multimedia systems«?
- Most engines support use of quotes to convey phrases
- Option 1: **Biword-index**
  - Use each sequential pair of terms as a combined term
  - Friends, Romans, Countrymen = [friends romans] [romans countrymen]
- Option 2: **Positional index**
  - For each term also store its positions in the document
  - Use positions to determine relations between words

Positional index example

Term »to« occurs **993427** times in the entire corpus. It occurs in documents {1,2,4,5,7}. In document 1 it occurs six times at places <7,18,33,72,86,231>.

to, 993427:
$\langle$ 1, 6: $\langle 7, 18, 33, 72, 86, 231\rangle$;
2, 5: $\langle 1, 17, 74, 222, 255\rangle$;
4, 5: $\langle 8, 16, 190, 429, 433\rangle$;
5, 2: $\langle 363, 367\rangle$;
7, 3: $\langle 13, 23, 191\rangle; \ldots\rangle$

be, 178239:
$\langle$ 1, 2: $\langle 17, 25\rangle$;
4, 5: $\langle 17, 191, 291, 430, 434\rangle$;
5, 3: $\langle 14, 19, 101\rangle; \ldots\rangle$

Positional index

- Proximity constraint (documents where word A and B are in distance x).
- Example:
  - **Query**: »to be or not to be«
  - **Terms**: to, be, or, not
  - Intersection of lists for »to« and »be«

to, 993427:
$\langle$ 1, 6: $\langle 7, 18, 33, 72, 86, 231 \rangle$;
2, 5: $\langle 1, 17, 74, 222, 255 \rangle$;
4, 5: $\langle 8, 16, 190, 429, 433 \rangle$;
5, 2: $\langle 363, 367 \rangle$;
7, 3: $\langle 13, 23, 191 \rangle$; … $\rangle$

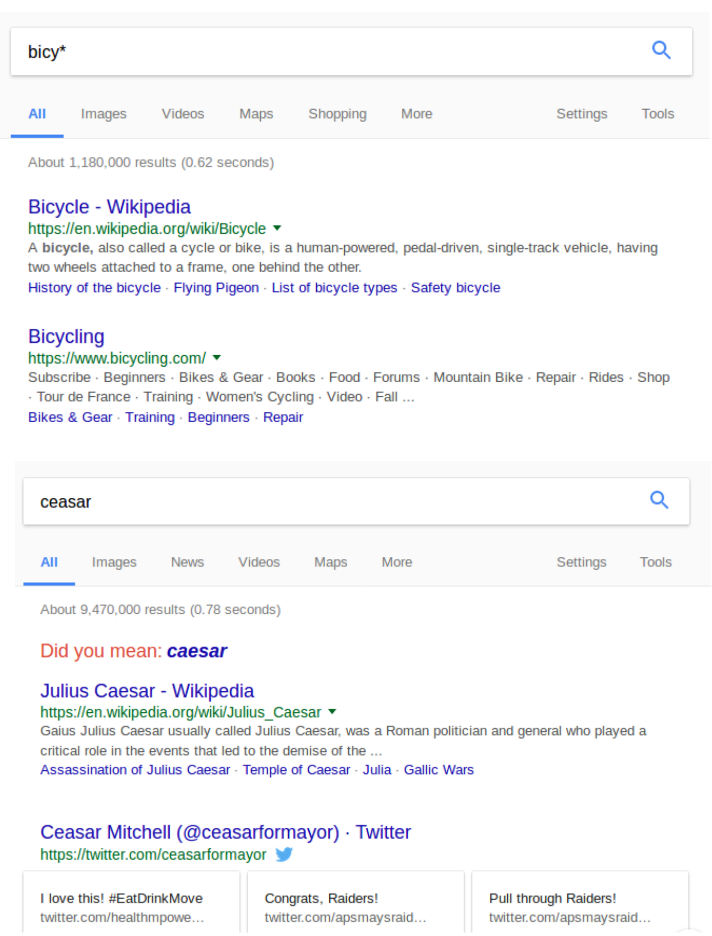
be, 178239:
$\langle$ 1, 2: $\langle 17, 25 \rangle$;
4, 5: $\langle 17, 191, 291, 430, 434 \rangle$;
5, 3: $\langle 14, 19, 101 \rangle$; … $\rangle$

Bi-word vs. positional index

- Bi-word index
  - More terms in index
  - Limited relations
- Positional index
  - Increased index size and complexity
  - Increased query time
- Combined approach:
  - Bi-word index for common phrases, e.g., »The Who«
  - Positional index for other terms

Tolerant retrieval

- Incomplete queries
  - Find words that begin with »vo«
  - Wildcard: *vo, vo*, pa*vo, …
- Typographical errors
  - »ceasar«
  - »gogle«

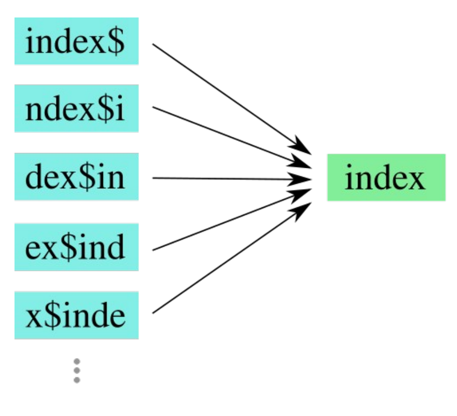


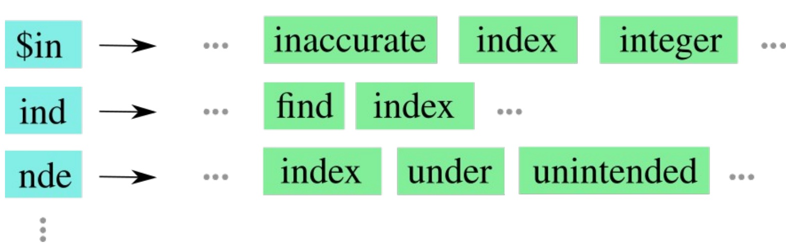
Wildcard queries

- Normal queries use hash table, which is not suitable for wildcard queries
- Use data-structures that order terms
  - Trees: order terms that all terms in a branch start with the same prefix
  - Wildcard queries generates different possible words that are then processed in a classical way with inverse index
  - Permuterm, K-gram index - significant increase in dictionary size
  - Boolean processing of wildcard queries is slow in general

Permuterm and K-gram indexing

- Permuterm
  - All shifts of word
  - Special stop sign
  - Large index
- K-gram
  - All sub-strings of length K
  - Special signs for start and stop
  - Brute-force post-processing





Typographical errors

- Example: »Britian Spears« instead of »Britney Spears«
- Correcting errors:
  - For each term separately: »padna« = »panda«
  - Based on neighbor terms (context) »Flew form Heathrow«
- Suggest corrections for terms that are not in dictionary
- Offer the most likely:
  - String distance (Levenshtein)
  - Phonetic distance (Soundex)
- Optional: with multiple equal possibilities offer the one that users use most often

Ranking documents

- Boolean queries only determine if a documents matches the query or not
  - Can generate large number of document
  - Time-consuming to check all of them
  - Show more relevant documents first

Ranking with term frequency

- Document that includes a queried term multiple times should be more relevant than the rest

$$\mathrm{tf}_{t,d} = \#\ \text{occurences of term}\ t\ \text{in document}\ d$$

- Bag-of-words model
- Problem - no context:
  - »Mary is quicker than John« equals to »John is quicker than Marry«

Problems of term frequency

- Some terms do not discriminate between documents because they occur in all of them
  - Corpus of documents in automotive industry will include term »auto« a lot
- Reduce weights of frequent terms – document frequency

$$\mathrm{df}_t = \#\text{ of document in which term } t \text{ appears at least once}$$

- Inverse document frequency:

$$\mathrm{idf}_t = \log_{10} \frac{N}{\mathrm{df}_t}$$

  - N – number of all documents in corpus
  - idf is low for frequent terms and high for scarce ones

Composite weights

- Term weight computed as : $\text{tf-idf}_{t,d} = \text{tf}_{t,d} \cdot \text{idf}_t = \text{tf}_{t,d} \log_{10} \frac{N}{\text{df}_t}$
- Weight is:
  - High: if t is frequent only in a small number of documents
  - Low: if t is rare or occurs in many documents
  - Very low: if term t occurs in almost all documents
- Compute document weight for query q

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

Ranking example

We have corpus of N=100 documents, in which we search for query q=»white pig«. We know in advance that »white« occurs in ten documents and that it occurs in document d1 five times. We also know that the word »pig« occurs in fifty documents and that it occurs in document d1 three times. Compute ranking weight of document d1 for query q.

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \cdot \text{idf}_t = \text{tf}_{t,d} \log_{10} \frac{N}{\text{df}_t}$$

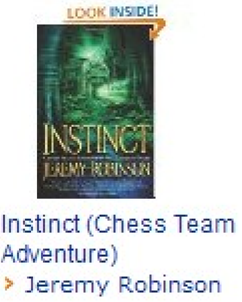
$$\text{Score}(q,d) = \sum_{t \in q} \text{tf-idf}_{t,d}$$

Document similarity

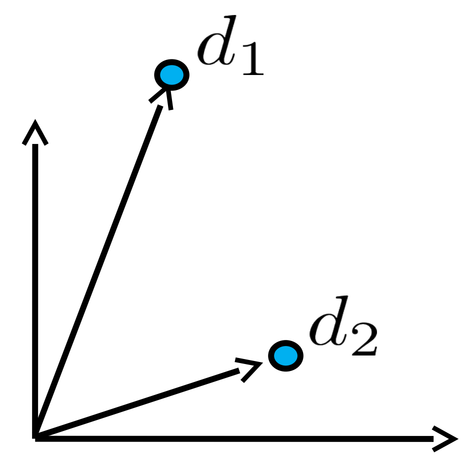
Document can be written as a vector of weights for all terms in the dictionary (similar to a histogram)

$$V(d) = [\text{tf-idf}_{t_1,d}, \text{tf-idf}_{t_2,d}, \ldots, \text{tf-idf}_{t_M,d}]$$

Customers Who Bought This Item Also Bought



Vector space



- Similarity as dot product in vector space

$$V(d) = [\text{tf-idf}_{t_1,d}, \text{tf-idf}_{t_2,d}, \ldots, \text{tf-idf}_{t_M,d}]$$

$$\text{sim}(\mathrm{d}_1, d_2) = V(d_1) \cdot V(d_2)$$

- Vector normalization
  - Longer vectors will have larger norm
  - Longer documents are not more important
  - Euclidean normalization $\mathrm{v(d)} = \mathrm{V(d)}\ /\ \|V(d)\|$, where $\|V(d)\| = \sqrt{\sum_{i=1}^{M} x_i^2}$
  - Similarity interpreted as cosine of angle between vectors

Similarity example

- Left table shows tf values (not weighted with idf) for dictionary of three terms for three books: SaS, PaP, and WH.
- Which document is most similar to document SaS?

Term frequency

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6

Normalized term frequency

term	SaS	PaP	WH
affection	0.996	0.993	0.847
jealous	0.087	0.120	0.466
gossip	0.017	0	0.254

Matching query with documents

- Vector space can be used to rank documents
- Similarity of query q and document d: $score(q,d) = v(q) \cdot v(d)$
- Example: q = »jealous gossip«

$$V(q) = [0, 1, 1] \rightarrow v(q) = [0, 0.707, 0.707]$$

$$score(q, \mathrm{SaS}) = ?, score(q, \mathrm{PaP}) = ?, score(q, \mathrm{WH}) = ?$$

Weighting schemes

- Other interpretations of tf and idf also exist
  - Can be different for query and documents
  - Proportional to the tf and idf qualities

$$0, 1 \ \text{(binary)}$$
$$f_{t,d}$$
$$\log(1 + f_{t,d})$$

$$1 \ \text{(unary)}$$
$$\log \frac{N}{df_t}$$
$$\log \left(\frac{N}{1 + df_t} \right) + 1$$

Weighting vectors example

- Query q=»best car insurance« in corpus of documents (N=100000)

	Corpus		Query (unary-idf)		Document (tf-idf)		
term	**df**	**idf**	**tf**	**w**	**tf**	**w**	**nw**
auto	5000	1.3	0	0	1	1.3	0.14
best	50000	0.3	1	0.3	0	0	0
car	10000	1.0	1	1.0	1	2.0	0.21
insurance	100	3.0	1	3.0	2	6.0	0.32

appearances in query

appearances in document

$$\mathrm{score}(\mathrm{q},\ \mathrm{d}) = 0 + 0 + 0.21 + 0.96 = 1.17$$

Document retrieval algorithm

- Cosine similarity between query sample and all documents
- Order documents by similarity
- Return top K documents
- Weighting with tf-idf requires:
  - For each term also store its document frequency
  - For every term in every document store term frequency

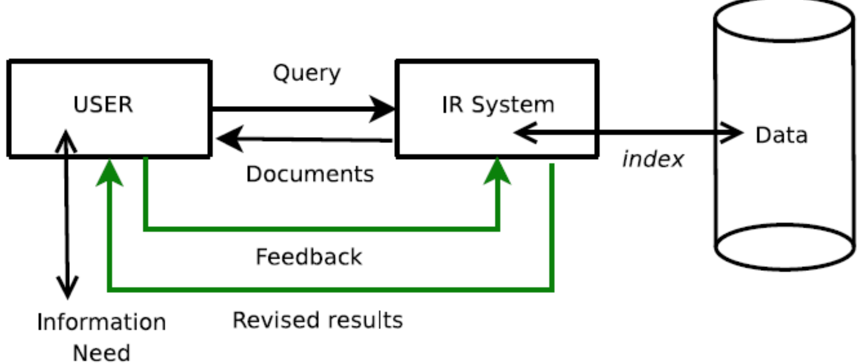
Feedback loops

- Multiple »words«, same concept
  - User does not know how to specify a specific enough query
  - Examples: »aircraft« vs. »plane«; »ship« vs. »boat«
- Global methods:
  - Expand query to as many possibilities with as many possible terms with error correction, synonyms, etc.
- Local methods:
  - Based on interaction between the user and the system
  - Relevance feedback

Relevance feedback

User reports information about relevance of individual results back to the system to improve the query

- Forming good queries is hard if the entire corpus is not known to the user
- Assessing individual documents is simple

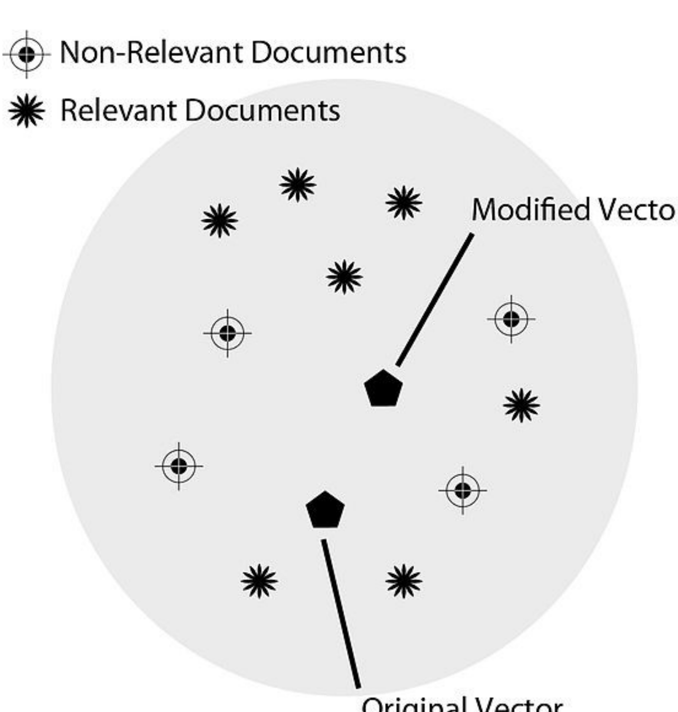


Rocchio algorithm

- Documents represented in vector space
- Known query and some relevant and irrelevant samples
- Formulate new query that is
  - Maximally similar to relevant results
  - Minimally similar to irrelevant results
- Use new query to retrieve better results

$$q_m = \alpha q_0 + \beta \frac{1}{|D_r|} \sum_{d_j \in D_r} d_j - \gamma \frac{1}{|D_n|} \sum_{d_j \in D_n} d_j$$

- q_0 - user query
- D_r - set of known relevant results
- D_n - set of known nonrelevant results
- α, β, γ - weights (e.g. $\alpha = 1, \beta = 0.75, \gamma = 0.15$)



Blind/pseudo relevance feedback

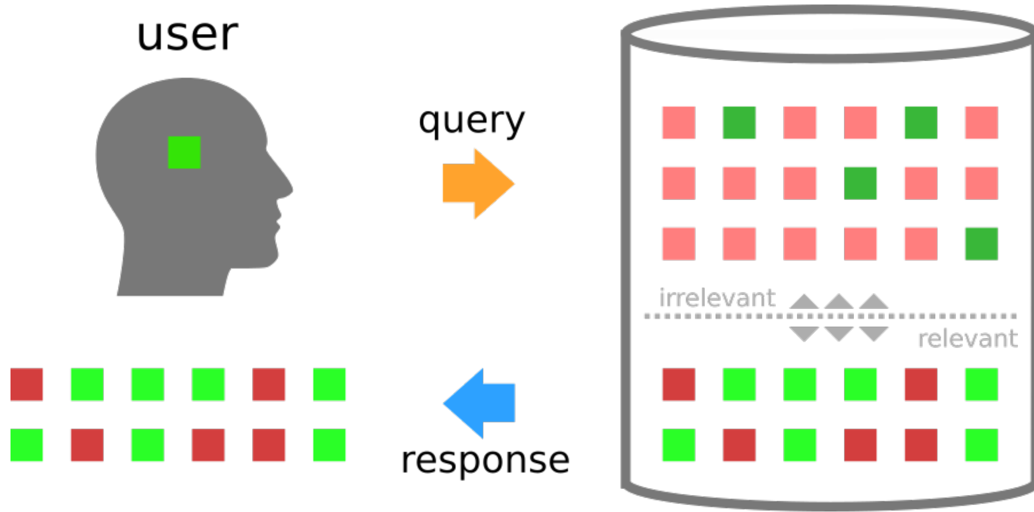
- Use default method to find most relevant documents
- Assume that K highest ranked documents are relevant
- Compute relevance feedback (Rocchio)
- Example TREC ad hoc task (Buckley et al. 1995)

	Precision at $k = 50$	
Term weighting	no RF	pseudo RF
lnc.ltc	64.2%	72.7%
Lnu.ltu	74.2%	87.0%

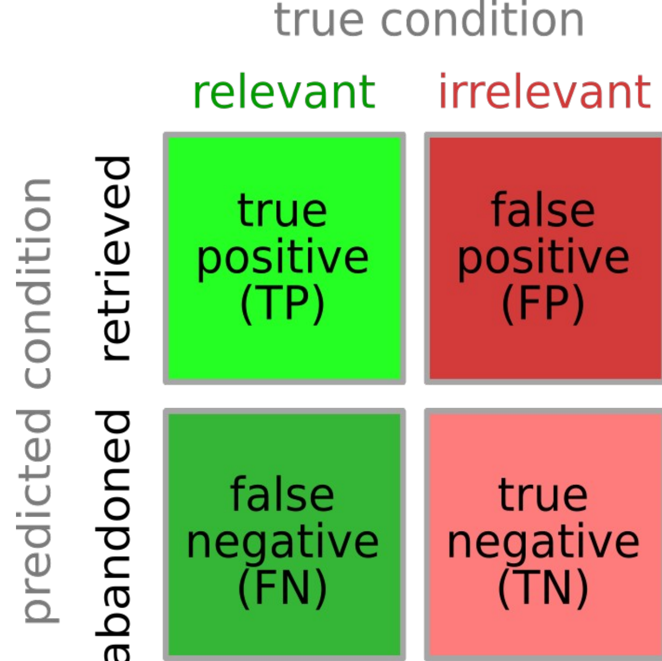
Objective retrieval performance

How many of the retrieved documents are relevant?

- Precision – percentage of relevant documents among retrieved documents
- Recall – percentage of returned relevant documents with respect to all relevant documents



Retrieval as classification



$$Precision = \frac{TP}{TP+FP}$$

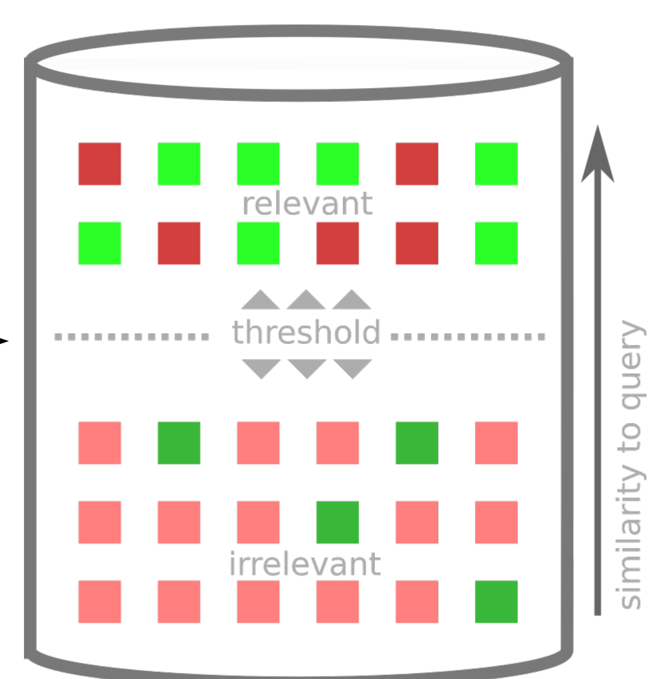
$$Recall = \frac{TP}{TP+FN}$$

Precision vs. recall

- Precision and Recall are related measures
  - Precision typically falls if the number of retrieved documents is increased
  - Recall increases if the number of retrieved documents is increased
- F-measure as a compromise $F = \frac{1}{\frac{\alpha}{Pr} + \frac{1-\alpha}{Re}}$
  - Typical weight $\alpha = 0.5 \rightarrow F = 2\frac{PrRe}{Pr+Re}$
- Higher value is better (maximum is 1)

Similarity threshold

- Decide which documents to return
  - Document similarity
  - Threshold $sim(q, d_i) > \epsilon$
- Depending on the threshold we get different precision and recall

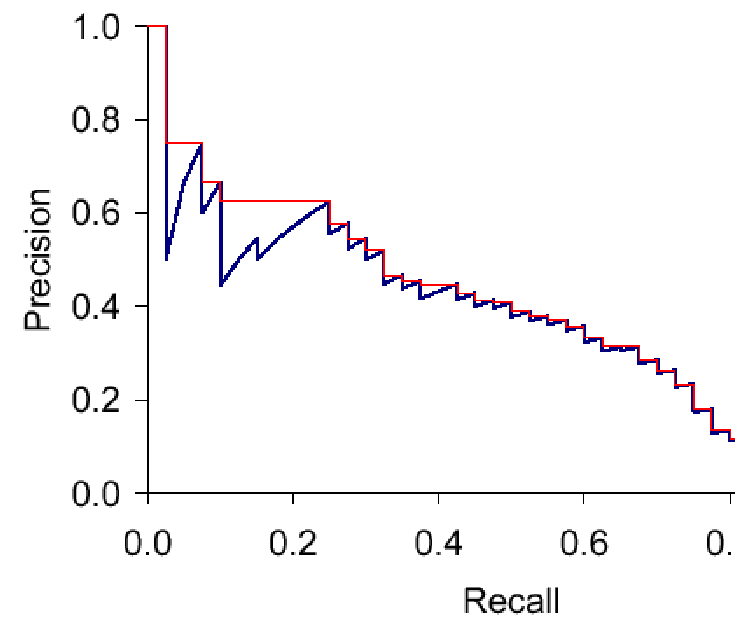


Plotting performance

- For each threshold we get a point in 2D space
- Visualize performance as plot for all thresholds
- Average Precision (AP) - Averaging over multiple thresholds (k)

$$AP = 1/|M| \sum_{k \in M} Precision(R_k)$$

- MAP (average of AP for multiple queries)



Retrieval performance analysis

- Dataset with ground-truth
  - Compute similarity for all documents
  - Compute TPR and FPR for threshold

$$TP_{rate} = \frac{TP}{TP+FN}$$

$$FP_{rate} = \frac{FP}{FP+TN}$$

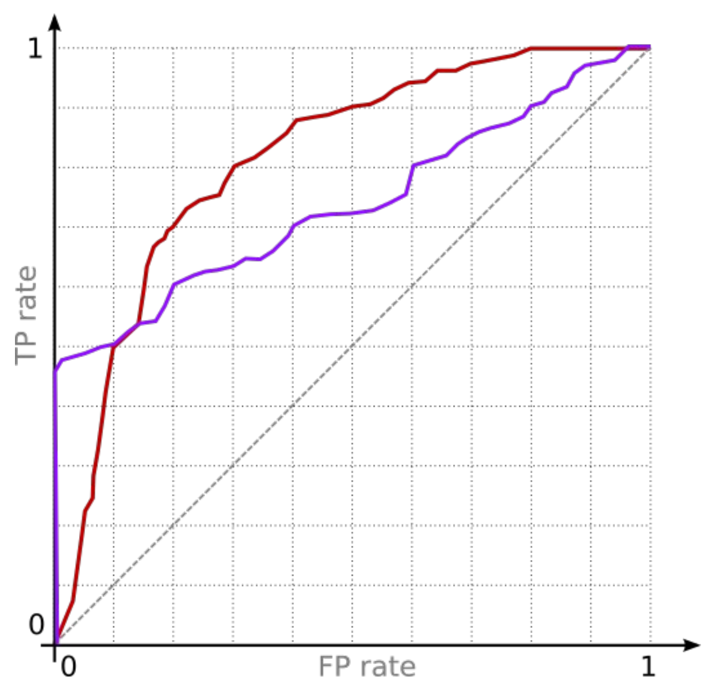
True condition:	1	1	0	1	0	1
Similarity:	1.0	0.2	0.1	0.8	0.9	0.8
For threshold **0.3**:	1 (TP)	0 (FN)	0 (TN)	1 (TP)	1 (FP)	1 (TP)

TP_{rate}=3/4=0.75

FP_{rate}=1/2=0.5

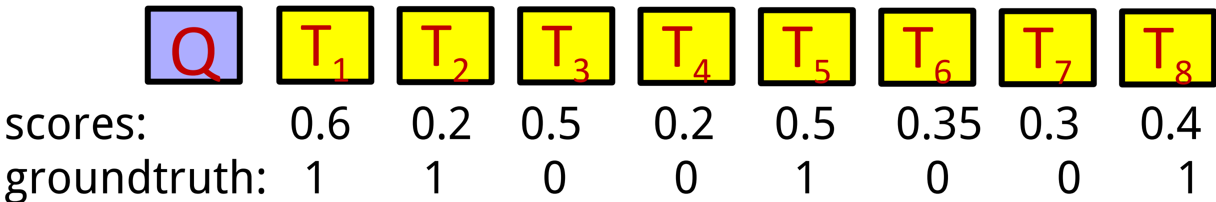
The ROC curve

- Receiver operating characteristic with respect to criterion (threshold)
  - True positive rate
  - False positive rate
- Interpretable measures
  - Distance to (0, 1)
  - Area under the curve (AUC)



ROC analysis example

- Documents are scored for relevance by their similarity to the query



- Calculate the ROC curve and determine optimal threshold
  - Sort documents by similarity
  - Set of unique similarity scores is threshold pool
  - For each threshold in pool calculate TPrate and FPrate
  - Each pair (FPrate, TPrate) is a point on a ROC curve
  - Select threshold that maximizes chosen criteria (e.g. point closest to (0,1))

Reading a ROC curve

- What is the percentage of retrieved relevant documents if we allow 20% of irrelevant documents in the result?
- What percentage of irrelevant results do we get if we want at least 90% of relevant documents in the results?