



Information retrieval

Structured vs. unstructured content

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

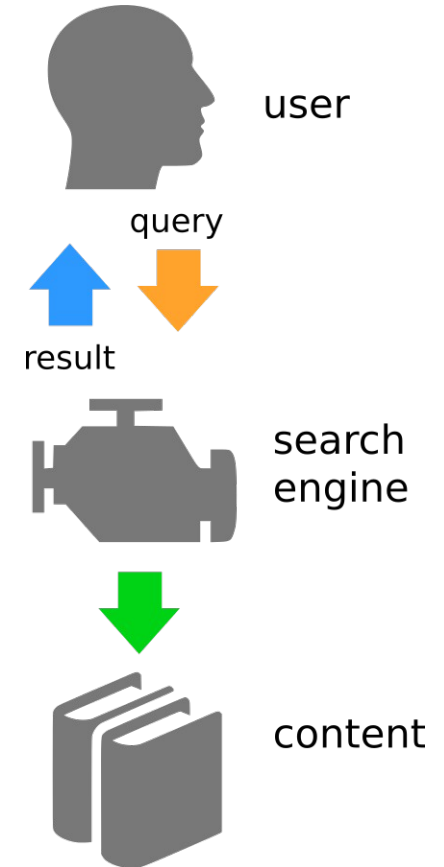


Overview

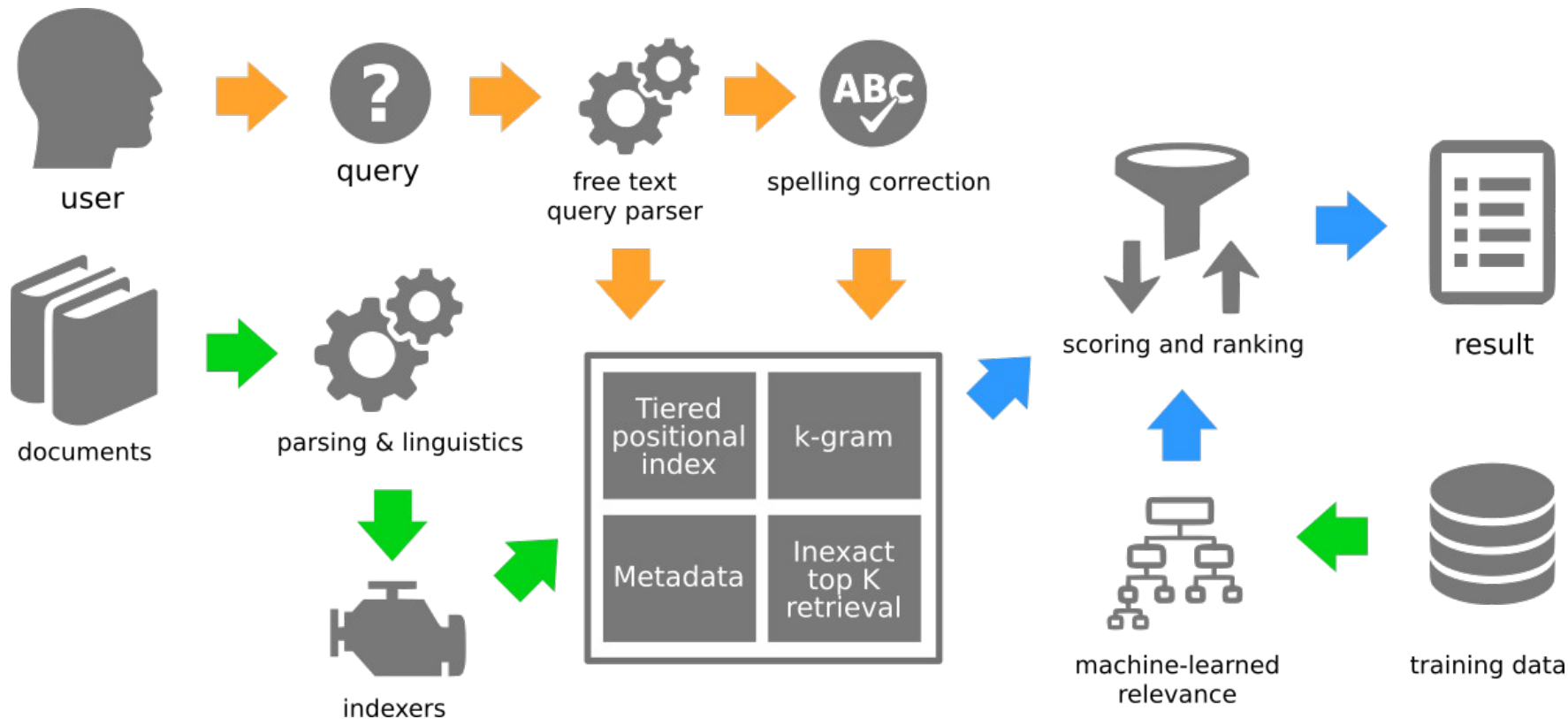
- Search using Boolean expressions
- Building search dictionary
- Approximate queries
- Ranking results
- Document similarity
- Feedback loops
- Retrieval systems evaluation

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

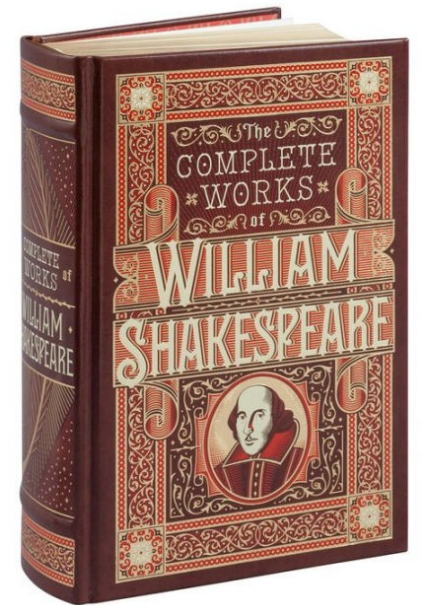


Search engine challenges

- 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

Documents 

	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	
...							

Terms 

Retrieval example

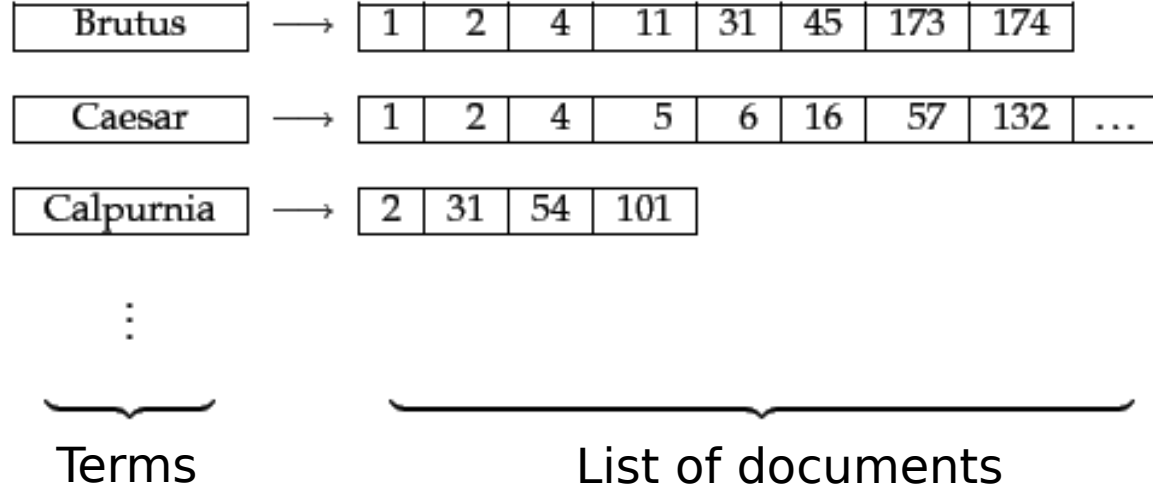
- Query: »Brutus AND Caesar AND NOT Calpurnia«
- Obtain binary vectors for all three terms, negate the last one and join them with AND:

	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

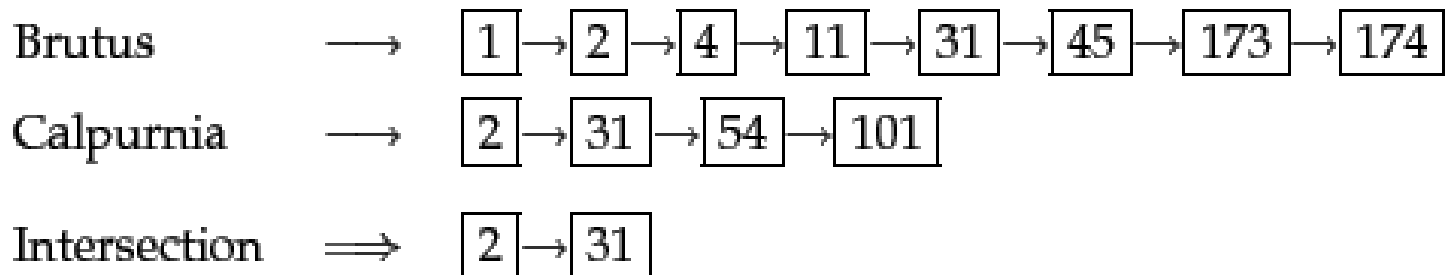
Inverted index

- 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)



Optimizing queries

- **AND query:** »Brutus AND Caesar AND Calpurnia«

- Reduce number of comparisons – start with two least frequent terms

Brutus	→	1	2	4	11	31	45		
Caesar	→	1	2	4	5	6	16	57	132
Calpurnia	→	2	31	54	101				

- 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

Decoding content

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

كِتَابٌ ← *
 un b ā t i k
 /kitābun/ 'a book'

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

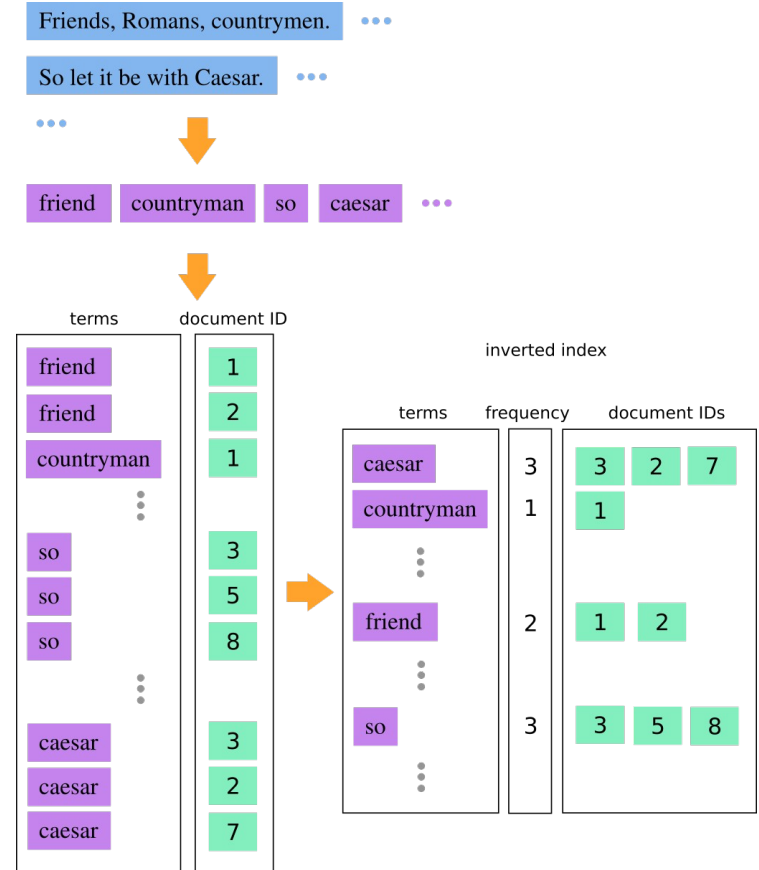
← → ← → ← START

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

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

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

Friends, Romans, countrymen, lend me your ears.



Friends Romans countrymen lend me your ears



Friends, Romans, countrymen, lend ears.



friend roman countryman lend ear

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)

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

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

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:

< 1, 6: <7, 18, 33, 72, 86, 231>;
2, 5: <1, 17, 74, 222, 255>;
4, 5: <8, 16, 190, 429, 433>;
5, 2: <363, 367>;
7, 3: <13, 23, 191>; ... >

be, 178239:

< 1, 2: <17, 25>;
4, 5: <17, 191, 291, 430, 434>;
5, 3: <14, 19, 101>; ... >

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:

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

be, 178239:

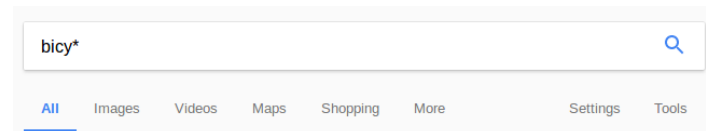
$\langle 1, 2: \langle 17, 25 \rangle;$
 $4, 5: \langle 17, 191, 291, 430, 434 \rangle;$
 $5, 3: \langle 14, 19, 101 \rangle; \dots \rangle$

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«



About 1,180,000 results (0.62 seconds)

[Bicycle - Wikipedia](#)

<https://en.wikipedia.org/wiki/Bicycle>

A bicycle, also called a cycle or bike, is a human-powered, pedal-driven, single-track vehicle, having two wheels attached to a frame, one behind the other.

[History of the bicycle](#) · [Flying Pigeon](#) · [List of bicycle types](#) · [Safety bicycle](#)

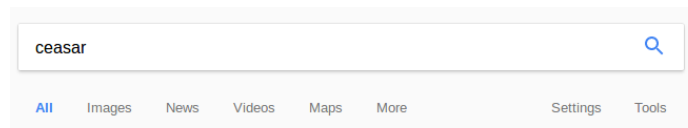
[Bicycling](#)

<https://www.bicycling.com/>

[Subscribe](#) · [Beginners](#) · [Bikes & Gear](#) · [Books](#) · [Food](#) · [Forums](#) · [Mountain Bike](#) · [Repair](#) · [Rides](#) · [Shop](#)

· [Tour de France](#) · [Training](#) · [Women's Cycling](#) · [Video](#) · [Fall ...](#)

[Bikes & Gear](#) · [Training](#) · [Beginners](#) · [Repair](#)



About 9,470,000 results (0.78 seconds)

Did you mean: [caesar](#)

[Julius Caesar - Wikipedia](#)

https://en.wikipedia.org/wiki/Julius_Caesar

Gaius Julius Caesar usually called Julius Caesar, was a Roman politician and general who played a critical role in the events that led to the demise of the ...

[Assassination of Julius Caesar](#) · [Temple of Caesar](#) · [Julia](#) · [Gallic Wars](#)

[Ceasar Mitchell \(@ceasarformayor\) · Twitter](#)

<https://twitter.com/ceasarformayor>

I love this! #EatDrinkMove
twitter.com/healthpowe...

Congrats, Raiders!
twitter.com/apsmaysraid...

Pull through Raiders!
twitter.com/apsmaysraid...

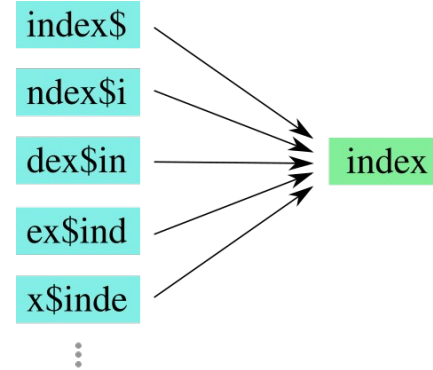
Wildcard queries

- 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

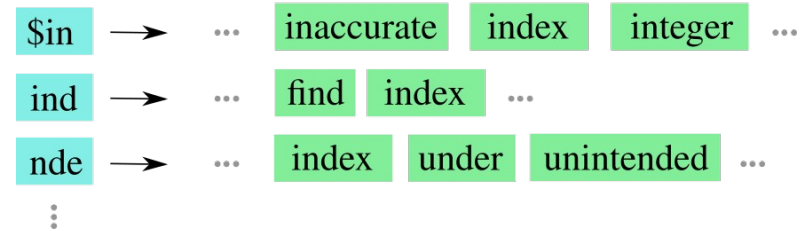
- Permuterm

- 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

$tf_{t,d} = \#$ occurrences of term t 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

$df_t = \#$ of document in which term t appears at least once

- Inverse document frequency: $idf_t = \log_{10} \frac{N}{df_t}$
 - 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 d_1 five times. We also know that the word »pig« occurs in fifty documents and that it occurs in document d_1 three times. Compute ranking weight of document d_1 for query q .

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

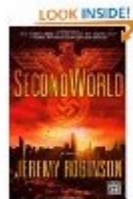
$$\text{Score}(q, d) = \sum_{t \in q} \text{tf-idf}_{t,d}$$

Document similarity

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

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

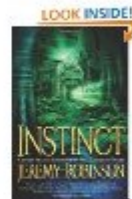
Customers Who Bought This Item Also Bought



SecondWorld
 > Jeremy Robinson
 ★★★★★☆ (40)



Threshold (Jack Sigler)
 > Jeremy Robinson
 ★★★★★☆ (28)



Instinct (Chess Team
 Adventure)
 > Jeremy Robinson

Vector space

- Similarity as dot product in vector space

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

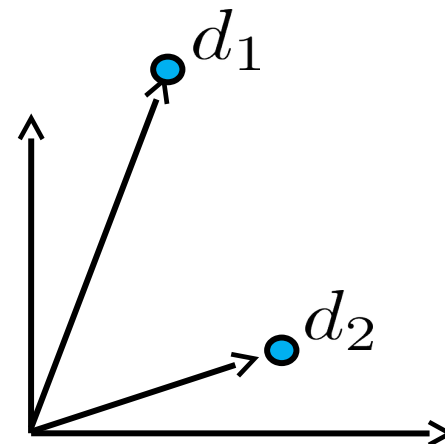
$$\text{sim}(d_1, d_2) = V(d_1) \cdot V(d_2)$$

- Vector normalization

- Longer vectors will have larger norm
- Longer documents are not more important

- Euclidean normalization $v(d) = V(d) / \|V(d)\|$, 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: $q = \text{«jealous gossip»}$

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

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

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

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

Weighting vectors example

- Query $q = \text{«best car insurance»}$ in corpus of documents ($N=100000$)

term	Corpus		Query (unary-idf)		Document (tf-idf)		
	df	idf	tf	w	tf	w	nw
auto	5000	1.3	0	0	1	1.3	0.21
best	50000	0.3	1	0.3	0	0	0
car	10000	1.0	1	1.0	1	1.0	0.16
insurance	100	3.0	1	3.0	2	6.0	0.96



 # appearances in query # appearances in document

$$\text{score}(q, d) = 0 + 0 + 0.16 + 2.89 = 3.04$$

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

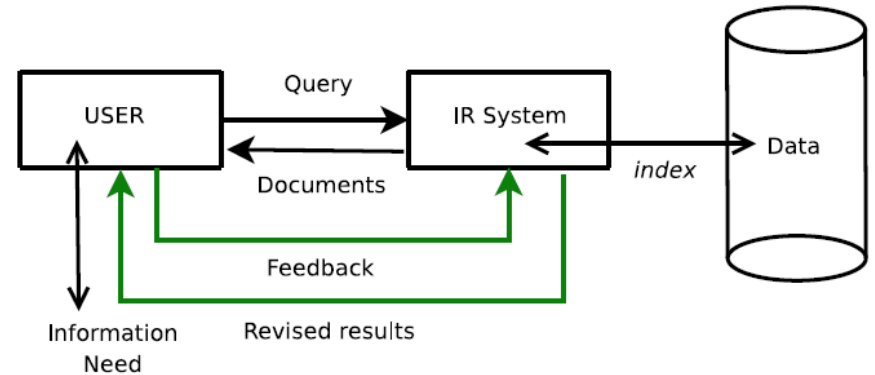
Feedback loops

- 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

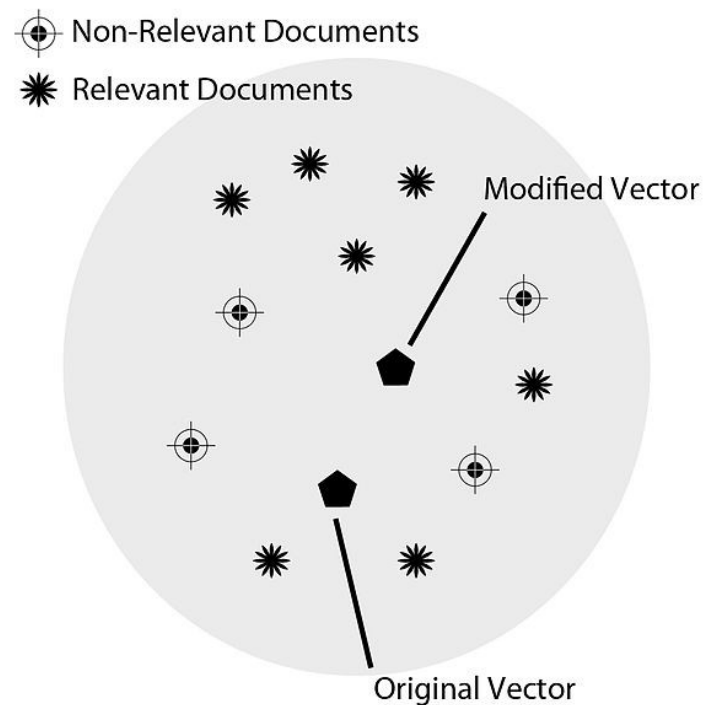


Rocchio algorithm

- 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

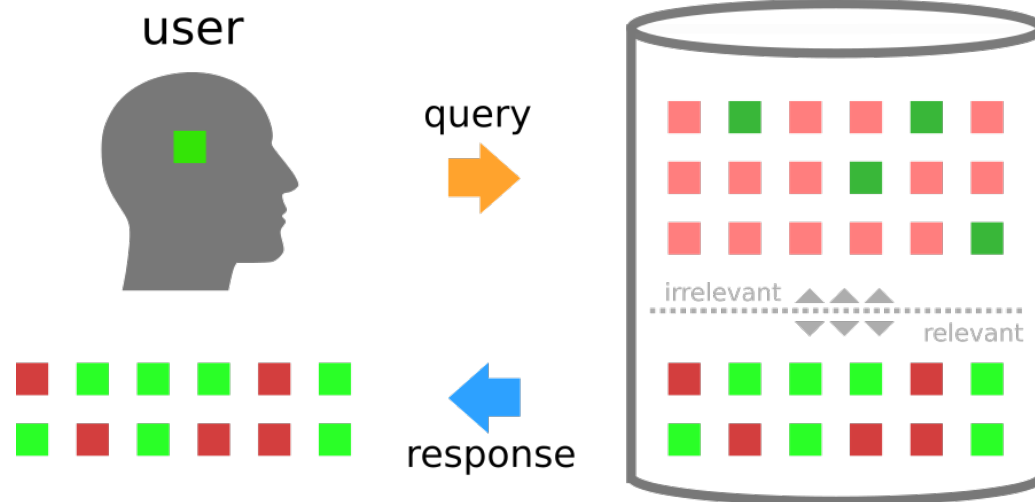
- Use default method to find most relevant documents
- Assume that K highest ranked documents are relevant
- Compute relevance feedback (Rocchio)
- Example TREC ad hoc task (Buckley et al. 1995)

Term weighting	Precision at $k = 50$	
	no RF	pseudo RF
Inc.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

		true condition	
		relevant	irrelevant
predicted condition	retrieved	<div style="background-color: #00FF00; padding: 10px; text-align: center;"> true positive (TP) </div>	<div style="background-color: #8B0000; padding: 10px; text-align: center;"> false positive (FP) </div>
	abandoned	<div style="background-color: #008000; padding: 10px; text-align: center;"> false negative (FN) </div>	<div style="background-color: #FF69B4; padding: 10px; text-align: center;"> true negative (TN) </div>

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

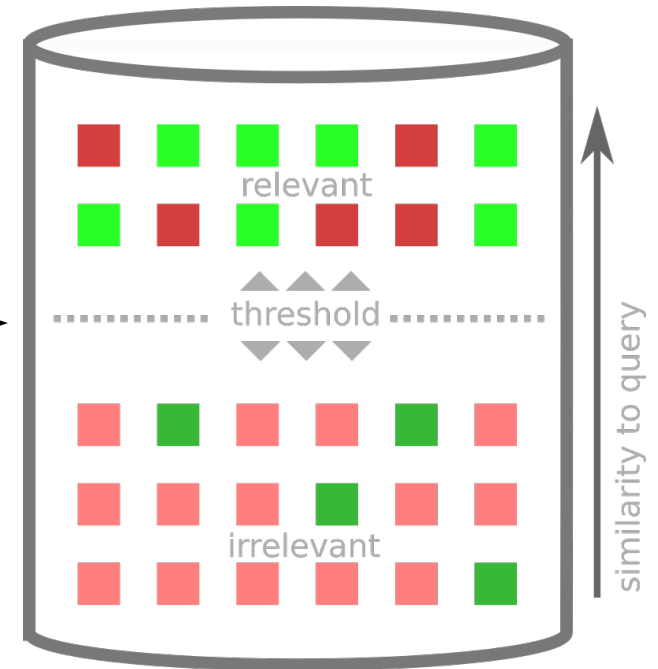
$$Recall = \frac{TP}{TP+FN}$$

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

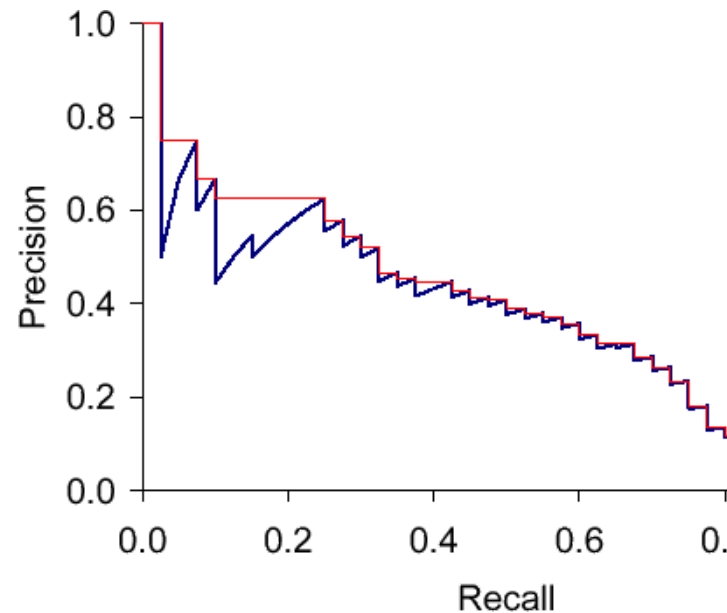


Plotting performance

- 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)

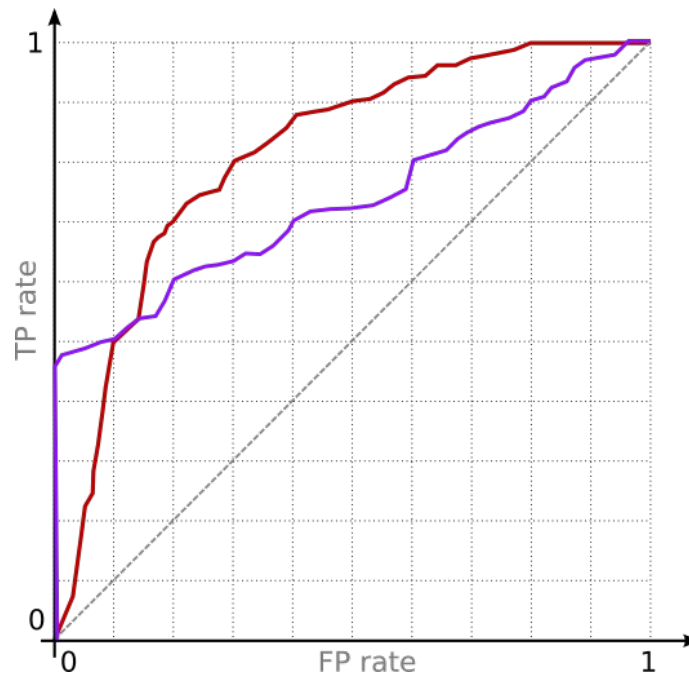


$$TP_{rate} = 3/4 = 0.75$$

$$FP_{rate} = 1/2 = 0.5$$

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

Q	T₁	T₂	T₃	T₄	T₅	T₆	T₇	T₈
scores:	0.6	0.2	0.5	0.2	0.5	0.35	0.3	0.4
groundtruth:	1	1	0	0	1	0	0	1

- 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?

