Information Retrieval(IR)

- Imdadullah Khan

Introduction

Information Retrieval (IR)

Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

Applications:

- Web search, E-mail search, Searching your laptop, Corporate knowledge bases, Legal information retrieval
- Search and communication are most popular uses of the computer
- Search is one of the the most fundamental computational problem

Information Retrieval: Problem Formulation

The Information Retrieval Problem

- Given a collection of documents
  Assume it is a static collection
- **Goal**: Retrieve documents with information that is *"relevant"* to the users need and helps the user complete a task

Information Retrieval: Key Issues

- How to "effectively" describe information resources (documents containing information)?
  - Feature extraction and Representation
  - Organization and Storage
  - Indexing
- How to find the "appropriate" information resources?
  - Accessing
  - Filtering
  - Retrieving
  - Ranking
  - Presenting

Information Retrieval: Keywords

- Automated IR systems designed in 60's to search news articles, scientific papers, patents, legal abstracts for keyword queries
- Keywords are short and inexpressive
- Problem of **synonymy** (different words with same meaning)
  - "soccer" would not retrieve documents containing the word "football"
- Problem of **polysemy** (same words with different meanings)
  - "Lie" could mean to lay down and to tell something untruthful
  - "fair" could be an event or could refer to skin color, or as in "just"
- Early IR system had two distinct features
  - Retrieval done by experts (reference librarians, patents attorneys)
  - Search space (knowledge base) were collections of documents written by experts
- Now everyone is an author and everyone is a searcher

Information Retrieval: Models

We briefly discuss two main information retrieval models

The goal is to set the stage for Web information retrieval, where there are many other issues

- Boolean Retrieval
- Ranked Retrieval

Boolean Retrieval

- Boolean retrieval model pose any query in the form of a Boolean expression of terms
- It combine sentences/documents with operators AND, OR, and NOT
- It views each document as a set of words/terms
- It is precise: document matches a condition or not
- It was the primary commercial retrieval tool for 3 decades
- Many search systems still in use are Boolean
  - Email, library catalog, Mac OS X Spotlight
- Main techniques used in Boolean retrieval are
  - Term Document Index Matrix
  - Inverted Index

Term-Document Incidence Matrix

- Store the collection of documents in Boolean form
- Term-document incidence matrix, M
- A row for each term and a column for each document
- $M(t, d)$ is 1 if doc d contains the term t and 0 otherwise

	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
...						

Term-Document Incidence Matrix

- **Example:** Retrieve Shakespeare's novel(s) that have words

  Brutus AND Caesar AND NOT Calpurnia

- Bruteforce: linear scan all docs for each query

- Bitwise AND of vectors for Brutus, Caesar and NOT Calpurnia

	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	1	0	1	1	1	1
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0
AND	1	0	0	1	0	0

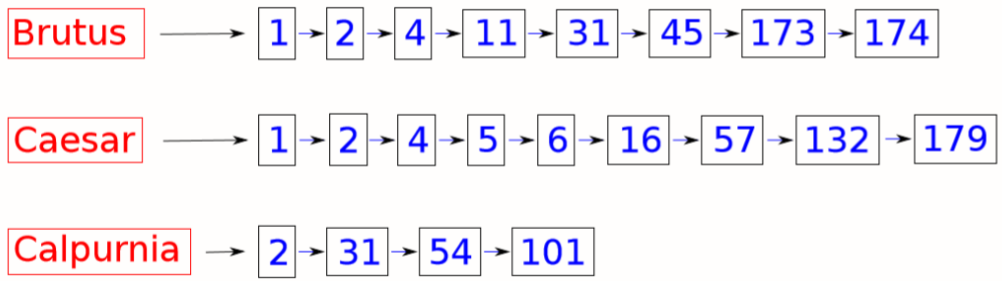
- Returns two documents, "Antony and Cleopatra" and "Hamlet"

Term-Document Incidence Matrix: Limitations

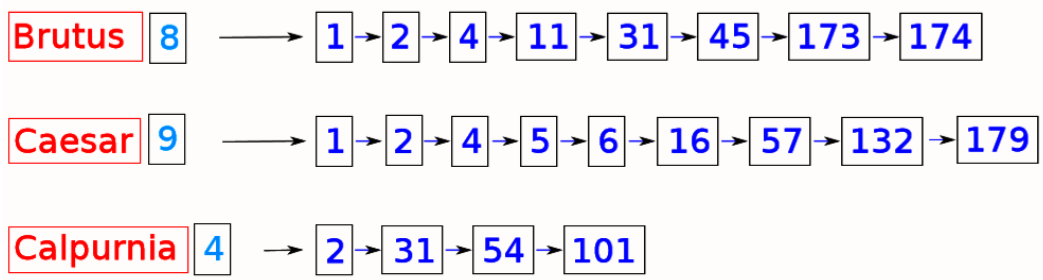
- Consider $N = 10^6$ documents, each with about 1000 terms (tokens)
- 10^9 tokens at avg 6 bytes per token $\rightarrow 6GB$
- Assume there are $|\Sigma| = 500,000$ distinct terms in the collection
- Size of incidence matrix is then $500,000 \times 10^6$ (half a trillion bits)
- Generally, the term-document matrix is very sparse (contains no more than a billion 1's)

The Inverted Index

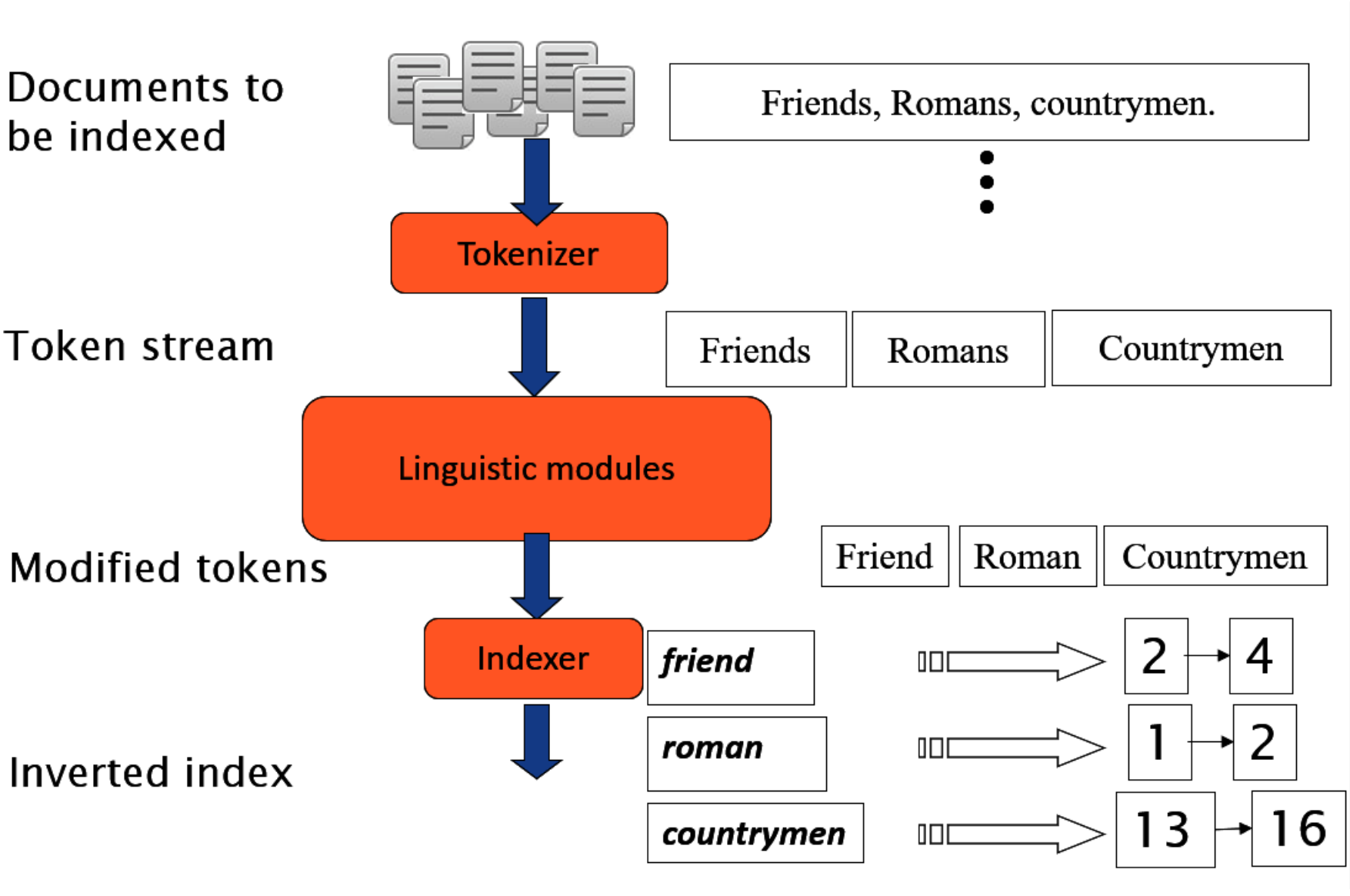
- Σ : set of all terms (also called lexicon, vocabulary)
- A postings list for each term in Σ
  - A list of document IDs in which the term occurs



- Sort each posting and the list of posting for optimal processing
- Save length of lists at header



The Inverted Index - Steps



The Inverted Index - Token sequence

- Sequence of (Modified token, Document ID) pairs

Doc 1

I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

Doc 2

So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

The Inverted Index - Sort

- Sort by terms
- then by doc-ID

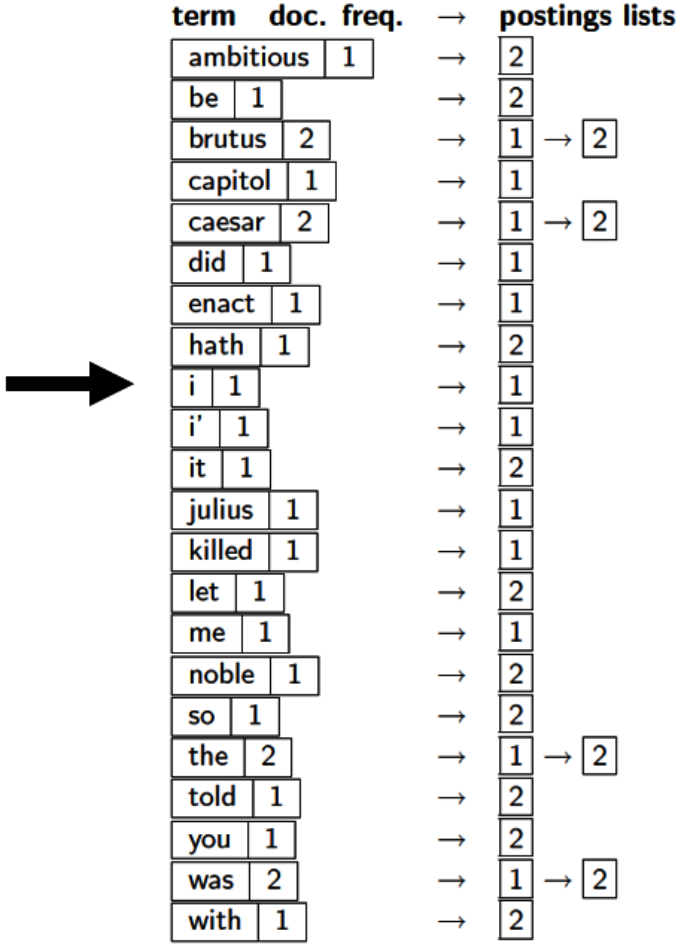
Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

The Inverted Index - Dictionary And Postings

- List document entries for each term
- Split into Dictionary and Postings
- Add number of docs information

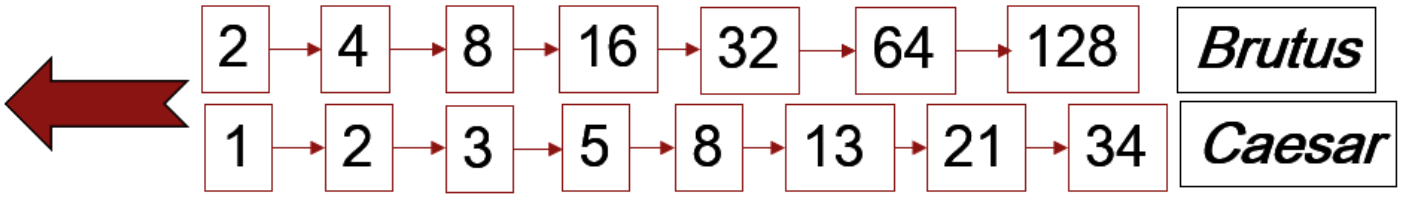
Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2



The Inverted Index - Query processing

Query: Brutus AND Caesar

- Locate Brutus in the dictionary and retrieve its postings
- Locate Caesar in the dictionary and retrieve its postings
- "Merge" the two postings (to get intersection)
- Walk through the two postings simultaneously, in time linear in the total number of postings entries
- Let the list lengths be x and y, merge takes $O(x + y)$ operations (since postings are sorted by docID)



The Inverted Index - More general merges

Query: Brutus AND NOT Caesar

- Complement posting would be very long
- Can we still compute intersection in $O(x + y)$?

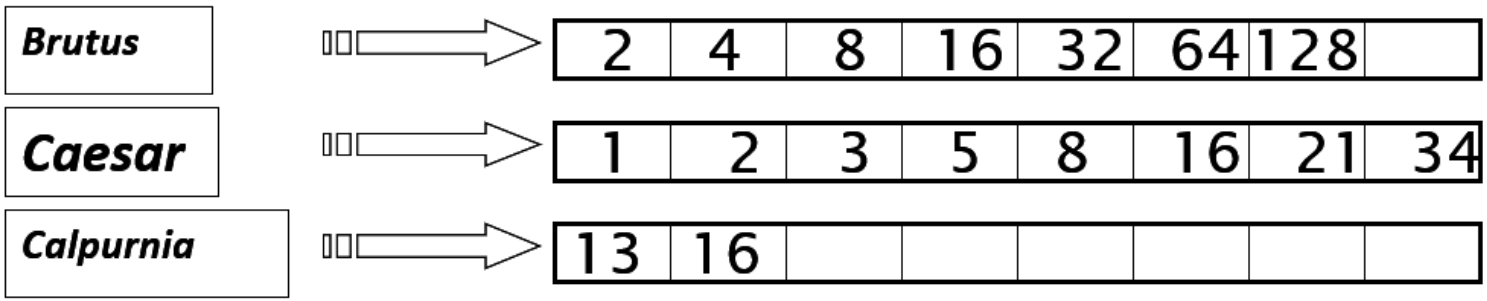
Query: (Brutus OR Caesar) AND NOT (Antony OR Cleopatra)

- Can we evaluate any set algebraic expression in "linear" time?

The Inverted Index - Query Optimization

Query: Brutus AND Calpurnia AND Caesar

- What is the best sequence of binary intersections to get $X \cap Y \cap Z$
- Process in order of increasing posting lengths
  - start with smallest pair of sets, then keep cutting further
  - this is why we kept document lengths in dictionary



Execute the query as (Calpurnia AND Brutus) AND Caesar

Inverted Index: Issues

Query: "Punjab University" as a phrase

- Documents containing "University of Punjab" are not a match
- Biword Indexes: Indexes bigrams in the text as phrases
  - Issues with Biword Indexes: Index blowup due to bigger dictionary
- Positional Indexes: also cater for context
  - In the postings, for each term store the position(s) where it appears
    <term, number of docs containing term;
    doc1: position1, position2 ;
    doc2: position1, position2 ;
    etc >
  - Issues with Positional Indexes:
    - A positional index is $2-4$ times as large as a non-positional index
    - Positional index size $35-50\%$ of volume of original text

Limitations of Boolean Retrieval

- Exact matching may retrieve too few (~ 0) or too many documents (~ 1000)
- Difficult to come up with a manageable query
  - AND gives too few, OR gives too many
- The burden is on the user to formulate a good Boolean query
- All terms are equally weighted
- Output is not ordered in useful fashion
- Complex query syntax (too much work for non-experts)

Ranked Retrieval

- Ranks documents by relevance to the query
- Return top k documents by relevance
- Allow for free text queries: Rather than considering query language of operators and expressions, it consider words of human language
- The size of the result is not an issue ($k \sim 10, 20, 100$)

Ranked Retrieval

Note scan pattern:

Page 3:
- Result 1
- Result 2
- Result 3
- Result 4
- Result 3
- Result 2
- Result 4
- Result 5
- Result 6 <click>

Q: Why do this?

A: What's learned later influences judgment of earlier content.

Scoring as Basis of Ranked Retrieval

- How to rank-order documents in collection with respect to a query?
- Assign a relevance score - say in [0,1] - to each document
- e.g query with one term : If the term does not occur in the document, score is 0. The more frequent the query term in document; the higher the score

Scoring Methods for Ranked Retrieval

Different measures to calculate relevance/similarity in ranked retrieval

- Exact Match Models
  - Unranked Boolean retrieval model
  - Ranked Boolean retrieval model (based on set/bag of words model)
- Best Match Model (Vector Space Model)
  - TF-IDF (and Cosine Similarity)

Vector Space Model

- Best-match models 'compute' the degree to which a document is relevant to a query
- It is expressed as $sim(q, d)$ (similarity between query and document)
- How to compute the similarity between q and d ??
- Vector space model represent both documents and queries by real vectors
- Similarity is evaluated in the vector-space
- e.g. cosine similarity between the vector representation of query and document

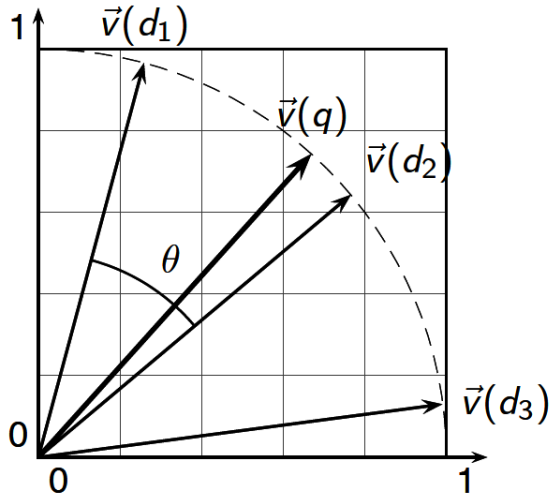
Summary – Vector Space Model Ranking

- Represent the query by the TF-IDF vector
- Represent each document by the TF-IDF vector
- Compute cosine similarity between the query and document vectors
- Rank documents with respect to the query by cosine similarity
- Return top k to the user

Vector Space Model

$$cos(\mathbf{q}, \mathbf{d}) = \frac{\mathbf{q}.\mathbf{d}}{|\mathbf{q}||\mathbf{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2 \sum_{i=1}^{|V|} d_i^2}}$$

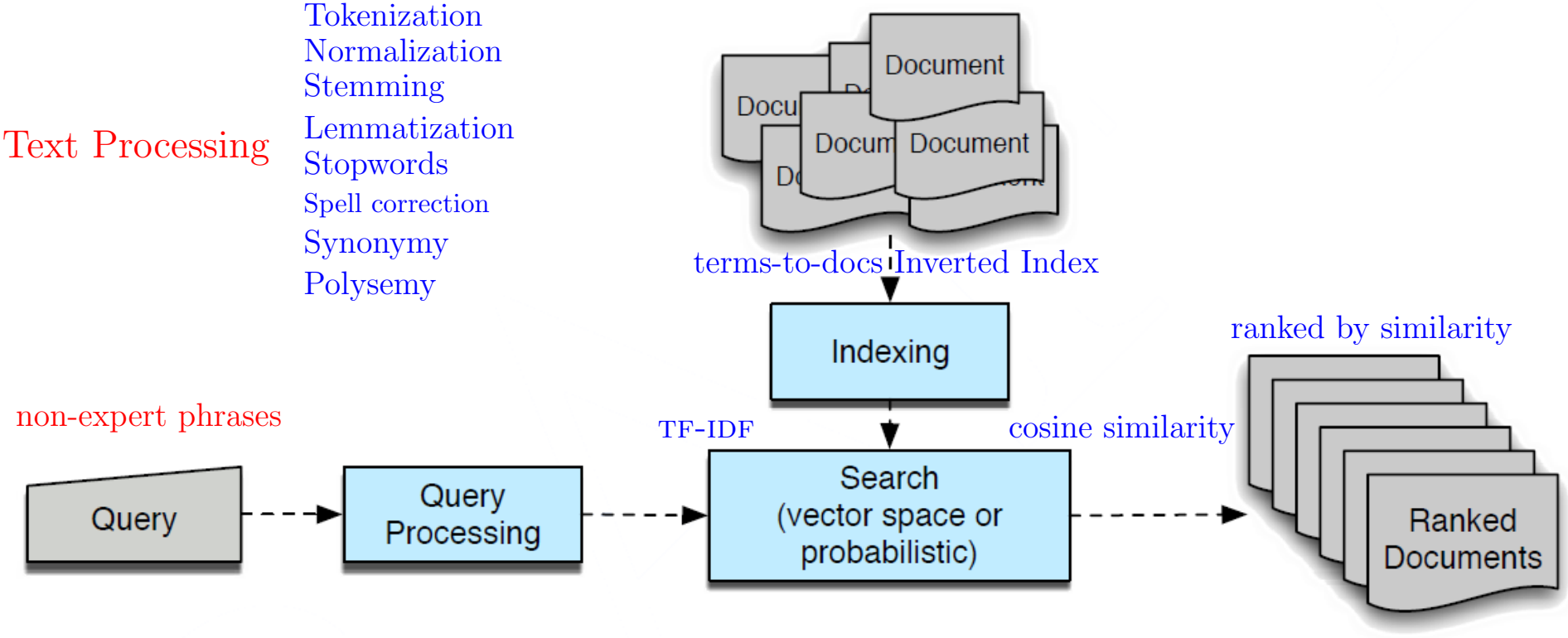
- q_i is the TF-IDF weight of term i in the the query
- d_i is the TF-IDF weight of the term i in the document



Information Retrieval

Text Processing

- Tokenization
- Normalization
- Stemming
- Lemmatization
- Stopwords
- Spell correction
- Synonymy
- Polysemy



Vector Space Model -Example

- **Query**: "best car insurance"
- **Document**: "car insurance auto insurance"

word	query					document				product
	tf-raw	tf-wght	df	idf	weight	tf-raw	tf-wght	weight	n'lized	
auto	0	0	5000	2.3	0	1	1	1	0.52	0
best	1	1	50000	1.3	1.3	0	0	0	0	0
car	1	1	10000	2.0	2.0	1	1	1	0.52	1.04
insurance	1	1	1000	3.0	3.0	2	1.3	1.3	0.68	2.04

Key to columns: tf-raw: raw (unweighted) term frequency, tf-wght: logarithmically weighted term frequency, df: document frequency, idf: inverse document frequency, weight: the final weight of the term in the query or document, n'lized: document weights after cosine normalization, product: the product of final query weight and final document weight