Information Retrieval

Grigorios Loukides
Email: grigorios.loukides@kcl.ac.uk

Session Objectives

In this session, you will learn:

- Information Retrieval (IR) Concepts
- Types of Queries in IR Systems
- Indexing
- Ranking Results
- Text Preprocessing

Information Retrieval

Information Retrieval

Information retrieval deals with the problems of storing, indexing, and retrieving (searching) such information to satisfy the needs of users

- User's information need expressed as a free-form search request

Data Retrieval vs. Information Retrieval

Databases

- Structured data
- Schema driven
- Relational (or object, hierarchical, and network) model is predominant
- Structured query model
- Rich metadata operations
- Query returns data
- Results are based on exact matching (always correct)

IR Systems

- Unstructured data
- No fixed schema; various data models (e.g., vector space model)
- Free-form query models
- Rich data operations
- Search request returns list or pointers to documents
- Results are based on approximate matching and measures of effectiveness (may be imprecise and ranked)

IR Pipeline

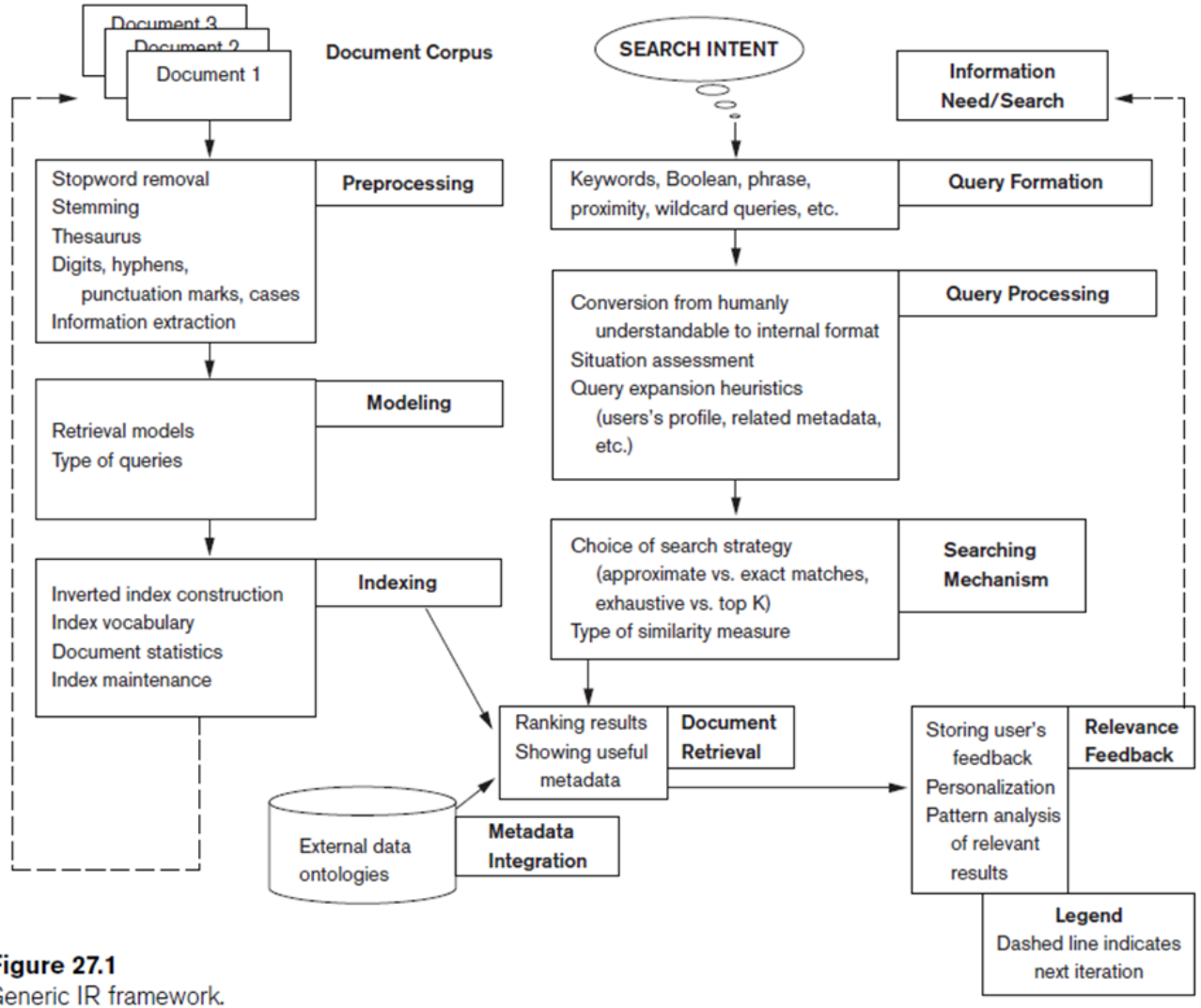


Figure 27.1
Generic IR framework.

Example of Information Retrieval Task

Suppose you wanted to determine which plays of Shakespeare contain the words: `Brutus AND Caesar AND NOT Calpurnia`

- This is an example of a boolean retrieval task
- SOLUTION: Linear scan through documents (grepping) (this is not effective for large document collections)

Definitions

- Document: whatever unit we have decided to build a retrieval system over (chapter, section, book)
- Word: a delimited string of characters as it appears in the text
- Term: a normalised word (case, morphology, spelling etc); an equivalence class of words
- Token: an instance of a word or term occurring in a document

Boolean Queries

Boolean retrieval model

- Queries are in the form of a Boolean expression of terms, that is, in which terms are combined with the operators AND, OR, and NOT
- The model views each document as just a set of words (no order or collocation)

Index

To avoid linearly scanning is possible to *index* the documents in advance:

- For each document a play of Shakespeare's we record whether it contains each word out of all the words (Shakespeare used 32.000 different words)
- The result is a binary term-document *incidence matrix*
- *Terms* are the indexed units (they are usually words)

	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	
...							

Realistic Example

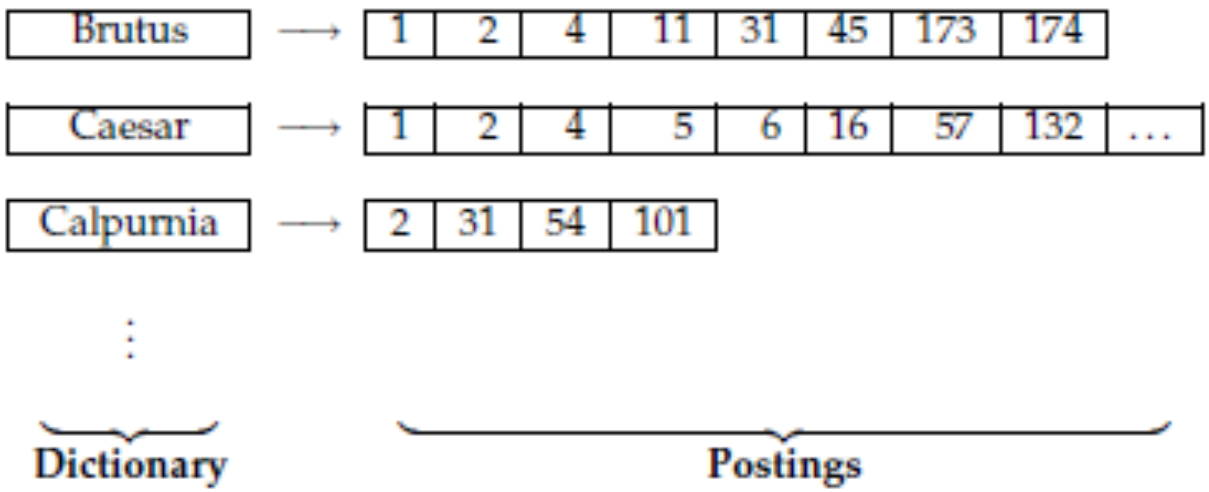
In a more realistic scenario we can have:

- 1M documents
- Each document 1000 words
- 500000 terms
- half-a-trillion 0's and 1's too many to fit in a computer's memory
- The matrix is extremely sparse (99.8% of the cells are zero)

A much better representation is to record only the things that do occur, that is, the 1 positions.

Inverted Index

- We keep a *dictionary* of terms
- For each term, we have a list that records which documents the term occurs in
- Each item in the list (named *posting*) records that a term appeared in a document (and, later, often, the positions in the document)
- The dictionary is sorted alphabetically and each postings list is sorted by document ID



Example: Building an Inverted Index

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:

Example: Building an Inverted Index

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

Example: Building an Inverted Index

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

$\Longrightarrow$

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

Example: Building an Inverted Index

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

⟹

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

⟹

term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
I	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

Exercise 1

Consider these documents:

Doc 1 breakthrough drug for schizophrenia

Doc 2 new schizophrenia drug

Doc 3 new approach for treatment of schizophrenia

Doc 4 new hopes for schizophrenia patients

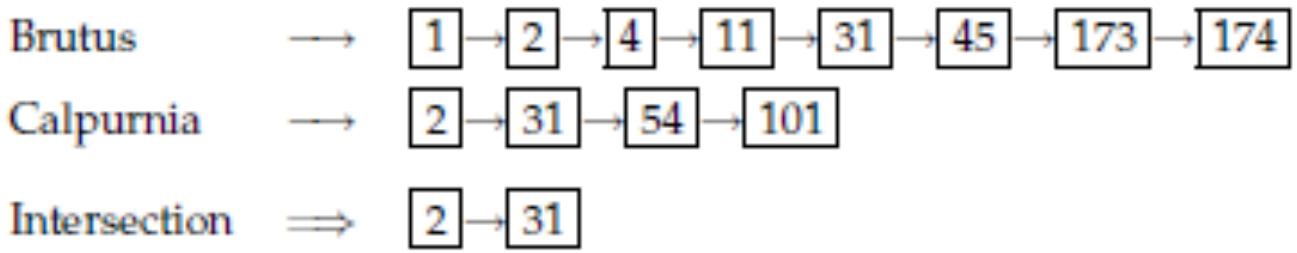
1. Draw the term-document incidence matrix for this document collection
2. Draw the inverted index representation for this collection,

Processing Boolean Queries:Simple Conjunctive Queries: Example

Consider processing the simple conjunctive query:

`Brutus AND Calpurnia`

1. Locate Brutus in the Dictionary
2. Retrieve its postings
3. Locate Calpurnia in the Dictionary
4. Retrieve its postings
5. Intersect the two postings lists



Intersection Algorithm


```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

Cost: $O(x + y)$ operations, x, y are the length of the posting lists

Formally, complexity of querying is $\Theta(N)$, where N is the number of documents in the collection

Query Optimization

Query optimization

Is the process of selecting how to organize the work of answering a query so that the least total amount of work needs to be done by the system

The standard heuristic is to process terms in order of increasing document frequency: if we start by intersecting the two smallest postings lists, then all intermediate results must be no bigger than the smallest postings list, and we are therefore likely to do the least amount of total work.

Exercise 2

For the queries below, can we still run through the intersection in time $O(x + y)$, where x and y are the lengths of the postings lists for Brutus and Caesar? If not, what can we achieve?

1. Brutus AND NOT Caesar
2. Brutus OR NOT Caesar

Phrase queries

- Phrase generally enclosed within double quotes:
  - "stanford university"
- about 10% of web queries are phrase queries
- Indexes no longer suffice

Positional indexes

Store, for each term, entries of the form:

```
<number of docs containing term;
doc1: position1, position2  ;
doc2: position1, position2  ;
etc.>
```

Processing Queries

"stanford university"

- Extract inverted index entries for each distinct term: stanford, university.
- Merge their doc:position lists to enumerate all positions with 'stanford university'.
- Same general method for conjunctive searches, but rather than simply checking that both terms are in a document, you also need to check that their positions of appearance in the document are compatible with the phrase query being evaluated

Exercise 3

Given the following fragment from an positional index,state how often the phrase "The undiscovered country" appears in each document:

```
the:
        3:34,38,55;
        5:12,16,25,44;
        7:67,87,90,101;
        10:33,39,45,62;

undiscovered:
        3:12,15,19;
        5:3,5,17,41,45,96;
        6:21,25,55,62;
        7:4,68,70,85,110;
        10:15,34,40,65,81;

country:
        3:22,26;
        5:18,46,52,65;
        7:5,69,91,105;
        8:32,42,65,93;
        10:32,44,75,83;
```

Intersection Complexity

- $\Theta(T)$, rather than $\Theta(N)$
  - T number of tokens in document collection
  - N number of documents in document collection
- Include frequent biwords as vocabulary terms in the index (Britney Spears)
  - Resolve all other phrases by positional intersection

Positional Index Size

- Need an entry for each occurrence, not just once per document
- Index size depends on average document size
  - Average web page has <1000 terms
  - SEC filings, books, even some epic poems ... easily 100,000 terms
  - Consider a term with frequency 0.1%

Document size	Postings	Entries in positional posting
1000	1	1
100,000	1	100

Positional Index Size

- A positional index is 2-4 as large as a non-positional index
- Positional index size 35-50% of volume of original text
- Caveat: all of this holds for "English-like" languages

Proximity queries

Accounts for how close within a record multiple terms should be to each other

LIMIT /3 STATUTE /3 FEDERAL /2 TORT

Here, /k means within k words of

- Positional indexes can be used for such queries;

Exercise 4

Consider the following fragment of a positional index with the format: word: document:<position, position,...>; document:<position, ...>

```
Gates: 1: <3>; 2: <6>; 3: <2,17>; 4: <1>;
IBM: 4: <3>; 7: <14>;
Microsoft: 1: <1>; 2: <1,21>; 3: <3>; 5: <16,22,51>;
```

The /k operator, word1 /k word2 finds occurrences of word1 within k words of word2 (In "w1 w2 w3", w3 is within 2 words from w1 (always left to right))

- Describe the set of documents that satisfy the query Gates /2 Microsoft.

Answer

- Describe the set of documents that satisfy the query Gates /2 Microsoft.

  None

Ranking Results

- Thus far, our queries have all been Boolean
  - Docs either match or not
- Good for expert users with precise understanding of their needs and the corpus
- Applications can consume 1000s of results
- Not good for (the majority of) users with poor Boolean formulation of their needs
- Most users don't want to wade through 1000's of results cf. use of web search engines

Scoring

We wish to return in order the documents most likely to be useful to the searcher

- How can we rank order the docs in the corpus with respect to a query?
- Assign a score for each doc on each query

Term Frequencies

- A document that mentions a query term more often has more to do with that query and should receive a higher score
- Term frequency: the number of occurrences of a term in a document. Sometimes normalized with total number of occurrences in the document.
  - Bag of words model

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

- This does not consider word order:

  John is quicker than Mary $\Leftrightarrow$ Mary is quicker than John

Term Frequencies (tf)

- Are all words in a document equally important?
- Consider the "ides of march" query
  - Julius Caesar has 5 occurrences of ides
  - No other play has ides
  - March occurs in over a dozen
  - All the plays contain of
  - By this scoring measure, the top-scoring play is likely to be the one with the most ofs
- Long docs are favoured because they are more likely to contain more query terms
- Can fix this to some extent by normalizing for document length
- But is raw tf the right measure?

Weighting term frequency

- What is the relative importance of
  - 0 vs. 1 occurrence of a term in a doc
  - 1 vs. 2 occurrences
  - 2 vs. 3 occurrences
- Unclear: while it seems that more is better, a lot isn't proportionally better than a few

Weighting should depend on the term overall

- Which of these tells you more about a doc?
  - 10 occurrences of hernia?
  - 10 occurrences of the?
- We need to consider term scarcity in collection (ides is rarer than of)
- Collection frequency (cf):total number of occurrences of a term in the collection
- Document frequency (df): Number of documents in the collection that contain a term

Word	cf	df
try	10422	8760
insurance	10440	3997

$tf \times idf$ term weights

- $tf \times idf$ (or $tf.idf$) measure combines:
  - term frequency (tf): number of times term occurs in document
  - inverse document frequency (idf)
    - measure of informativeness of a term: its rarity across the whole corpus
    - the most commonly used version is:

$$idf = \log \frac{N}{df}$$

      where N is the number of documents in a collection
    - it assigns to term t a weight in document d that is:
    1. highest when t occurs many times within a small number of documents;
    2. lower when the term occurs fewer times in a document, or occurs in many documents;
    3. lowest when the term occurs in virtually all documents

Exercise 5

$$idf = \log \frac{N}{df}$$

- What happens to the idf if a term appears in all documents?

Document Vector

- Each doc can now be viewed as a vector of $tf.idf$ values, one component for each term
- The score of a document d is the sum, over all query terms, of the $tf.idf$ weight of each term in d:

$$score(q, d) = \sum_{t \in q} tf.idf_{t,d}$$

Exercise 6

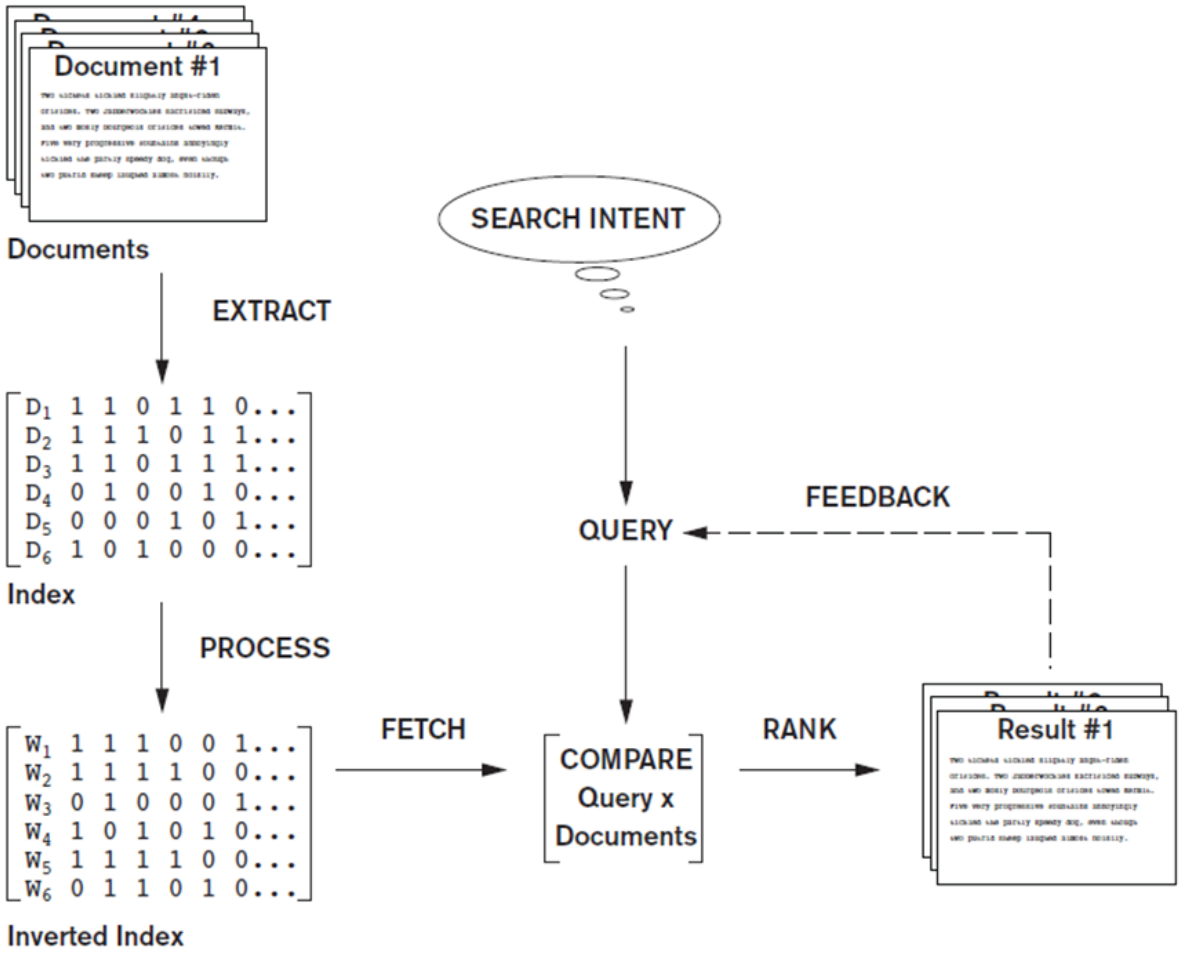
Assume we have a document collection that contains 20.000 documents and the word :

- 'Albert' appears in 300 documents
- 'Ludwigs' appears in 40 documents
- 'University' appears in 10.500 documents
- 'in' appears in 19.500 documents
- 'Freiburg' appears in 6.000 documents
- 'Germany' appears in 10.000 documents

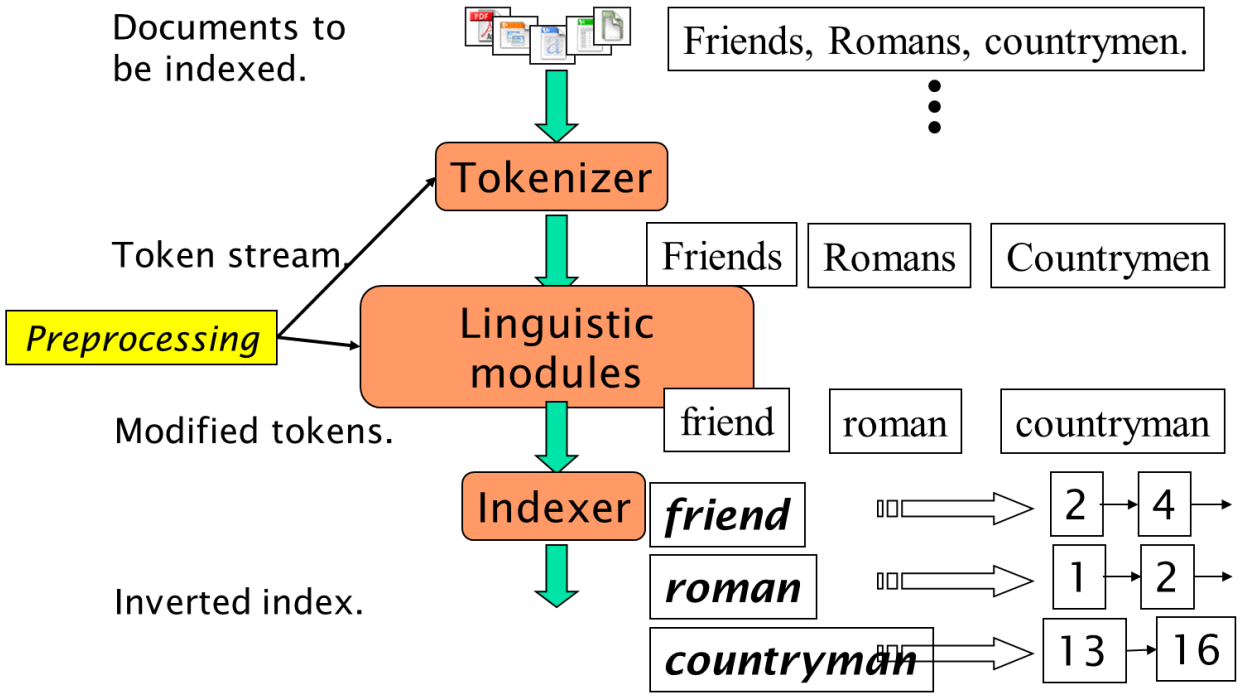
Calculate the TF*IDF vector for the following document:

- 'Albert-Ludwigs-University Freiburg, in Freiburg, Germany'

Information Retrieval



Preprocessing



Parsing a document

- What format is it in?
- What language is it in?
- What character set is in use?

All these questions can be solved by classifiers
But there are complications...

Tokenization

To build an inverted index, we need to get from

INPUT "Friends, Romans and Countrymen"

OUTPUT Tokens: Friend | Romans | Countrymen

But what are valid tokens to emit?

Issues in tokenization

- Finland's capital→ Finland? Finlands? Finland's?
- Hewlett-Packard → Hewlett and Packard as two tokens?
  - State-of-the-art: break up hyphenated sequence.
  - co-education ?
- San Francisco: one token or two? How do you decide it is one token?
- Numbers: 3/12/91, Mar. 12, 1991

Dropping common terms: stop words

Sometimes, some extremely common words (STOP WORDS) which would appear to be of little value in helping select documents matching a user need are excluded from the vocabulary entirely

a, an, and, are, as, at, be, by, for, from, has, he, in, is, it...

- Used to be standardly non-indexed in older IR systems.
- Length of practically used stoplists has shrunk over the years
- Most web search engines do index stop words

Normalization

- Need to "normalize" terms in indexed text as well as query terms into the same form:
  - We want to match U.S.A. and USA
- We most commonly implicitly define equivalence classes of terms
- Alternatively, we could do asymmetric expansion:

$$\text{window} \rightarrow \text{window, windows}$$
$$\text{windows} \rightarrow \text{Windows, windows, window}$$
$$\text{Windows} \rightarrow \text{Windows}$$

- Either at query time, or at index time
- More powerful, but less efficient

Thesauri: Handle synonyms and homonyms

- Hand-constructed equivalence classes
  - e.g., car = automobile
  - Rewrite to form equivalence classes
- Index such equivalences
  - When the document contains automobile, index it under car as well (usually, also vice-versa)
- Or expand query?
  - When the query contains automobile, look under car as well

Lemmatization & Stemming

For grammatical reasons, documents are going to use different forms of a word

organize, organizes, and organizing

Additionally, there are families of derivationally related words with similar meanings

democracy, democratic, and democratization

In many situations, it seems as if it would be useful for a search for one of these words to return documents that contain another word in the set

Lemmatization

- Reduce inflectional/variant forms to base form

  am, are, is $\rightarrow$ be

- Lemmatization implies doing "proper" reduction to dictionary headword form (the lemma)

Stemming

- Reduce terms to their "roots" before indexing
- "Stemming" suggest crude affix chopping language dependent

automate, automation, automatic $\rightarrow$ automat

IR Pipeline

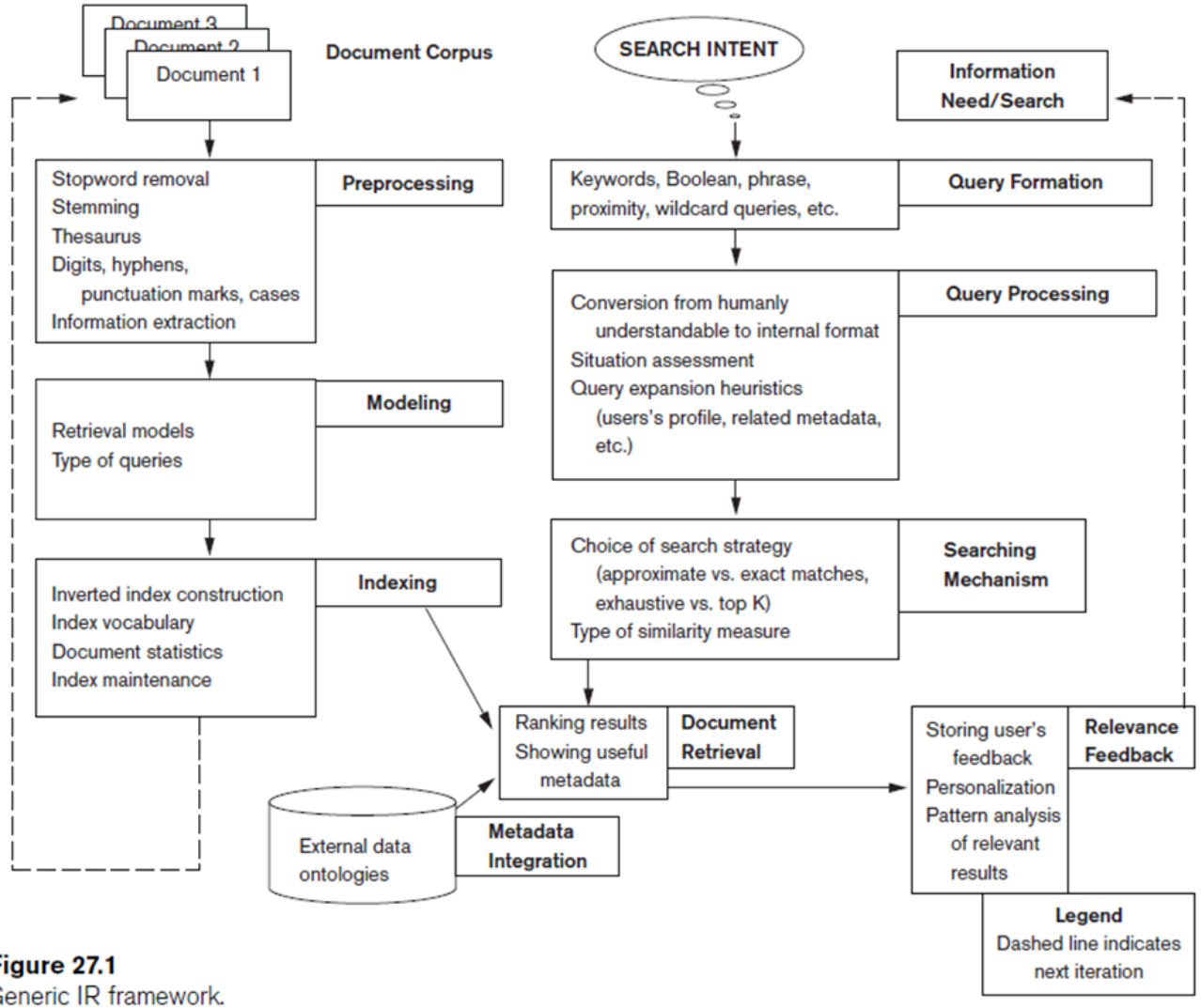


Figure 27.1
Generic IR framework.

Conclusion

In this session we have covered:

- IR main concepts
- Query Types
- Indexes
- Ranks
- Preprocessing

Suggested Readings

- Chapters 1, 2, and 6 on the book Introduction to Information Retrieval by Manning, Raghavan and Schtze
- Chapter 27 of Fundamentals Of Database Systems by Elmasri and Navathe.